UNIVERSITÀ DEGLI STUDI
DI SALERNO

UNIVERSITÀ DEGLI STUDI DI SALERNO
Faculty of Computer Science

A project report submitted for the course of Machine Learning

Fidelio: Developing a machine learning model for movie recommendation tasks

Supervisors:
Prof. Giuseppe Polese
Prof. Loredana Caruccio
Candidate:
Matricola: 0512119040
Nikolas Tullo

Anno Accademico 2025-2026

UNIVERSITÀ DEGLI STUDI DI SALERNO

ABSTRACT

Faculty of Computer Science
Dipartimento di Informatica

Link al github: `https://github.com/jacob14047/Fidelio`

The exponential growth of digital entertainment has made it increasingly difficult for users to navigate vast movie catalogs. In the context of the *Machine Learning* course, this project addresses the problem of information overload by designing and evaluaing an Hybrid Recommender System architectures. The experimental dataset was constructed by integrating interaction matrices and reviews from the MovieLens database and RottenTomatoes, with rich metadata harvested via the TMDb API.

To evaluate the proposed strategie, the following pipeline were implemented. The pipeline employs a stacking ensemble technique, where Content Encodings and Collaborative signals are fused and processed by a Gradient Boosting Regressor to predict user ratings. The system aims to mitigate common limitations such as the cold-start problem and data sparsity while achieving decent recommendation accuracy.

Contents

List of Figures

List of Tables

Chapter 1

Recommender Systems

1.1 Introduction

Recommender systems are intelligent information filtering systems designed to help users navigate the overwhelming abundance of digital content. In an era where Netflix hosts thousands of movies, Amazon lists millions of products, and Spotify offers tens of millions of songs, the ability to efficiently discover relevant content has become essential. These systems aim to predict what rating a given user would assign to a specific item, enabling personalized recommendations that match individual preferences.

The fundamental challenge addressed by recommender systems is the *information overload problem*: human cognitive capacity is limited, yet the volume of available choices continues to grow exponentially. Without intelligent filtering, users would spend excessive time searching for content, leading to decision fatigue and suboptimal choices. Recommender systems solve this by automatically analyzing patterns in user behavior, item characteristics, and collective preferences to surface the most relevant options.

This domain remains active and complex because individual tastes are not static—preferences evolve dynamically over time, influenced by trends, life events, and exposure to new content. Successfully modeling and anticipating these shifting preferences requires sophisticated algorithms that can adapt to changing user behavior while maintaining accuracy and relevance.

1.2 Real-World Applications

Recommender systems have become ubiquitous across digital platforms, fundamentally shaping how users discover and consume content. Their importance lies in their ability to dramatically improve user efficiency by reducing the time and cognitive effort required for decision-making.

1.2.1 Entertainment and Media

Video Streaming: Netflix employs advanced collaborative filtering and deep learning algorithms to recommend movies and TV shows based on viewing history and implicit feedback (pause behavior, rewatch patterns, time-of-day viewing). Similar platforms like Hotstar, Sony LIV, Voot, and ALTBalaji utilize recommendation technology to retain user engagement and reduce churn.

1.2.2 E-Commerce and Retail

Product Recommendations: Amazon, the world's largest online retailer, attributes a significant portion of its revenue to its recommendation engine, which suggests products based on browsing behavior, purchase history, and collaborative signals from similar users ("Customers who bought this also bought..."). Competitors like Flipkart, eBay, Myntra, and ShopClues have adopted similar systems, highlighting the strategic importance of recommendations in driving sales.

1.3 Problem Formulation

From an Artificial Intelligence perspective, we can formally model our recommender system as a *rational agent*—an entity that perceives its environment through sensors and acts through actuators to maximize a performance measure. This abstraction, known as the PEAS framework (Performance measure, Environment, Actuators, Sensors), provides a rigorous foundation for analyzing the system's objectives and constraints.

Performance Measure: The criteria used to evaluate the success of the recommendation agent.

- *Predictive Accuracy:* Minimizing the error between predicted ratings $\hat{r}_{ui}$ and actual user ratings r_{ui}, measured via **RMSE** (Root Mean Squared Error) and **MAE** (Mean Absolute Error). Lower error indicates more accurate predictions of user preferences.
- *Ranking Quality:* The ability to correctly identify and order the top-k items that the user will prefer, evaluated using precision and recall metrics. This matters more than exact rating prediction in many practical scenarios.
- *Catalog Coverage:* The percentage of items in the catalog that the system is capable of recommending. High coverage ensures diverse recommendations and addresses the "Long Tail" problem where niche items receive no recommendations.

Environment: The operational context in which the agent functions.

- *Item Catalog:* A collection of 85,235 movies from the **TMDB** dataset, each characterized by rich metadata including genres, cast, crew, keywords, and user ratings.
- *User Base:* The set of active users from the **MovieLens** dataset who interact with the system through explicit ratings and implicit behavioral signals merged with the analogous parameters obtained from the **RottenTomatoes** dataset.
- *Environment Properties:* The environment is **partially observable** (we cannot directly observe a user's true preferences or internal state), **stochastic** (user behavior contains inherent randomness), and **dynamic** (preferences evolve over time, and new content continuously enters the catalog).

Actuators: The actions the agent can perform to affect the environment and influence user experience.

- *Recommendation Generation:* Outputting a ranked list of K movies tailored to a specific user's predicted preferences.
- *Rating Prediction:* Estimating a scalar value $\hat{r}_{ui}$ (e.g., 1-5 stars) for a specific user-item pair (u, i) that has not yet been observed.

Sensors: The input mechanisms used to perceive the state of the environment and gather information about users and items.

- *Explicit Feedback:* Quantitative ratings (0.5 to 5.0 stars) provided voluntarily by users, recorded in the `ratings.csv` file. These represent direct expressions of preference.
- *Content Metadata:* Textual and categorical features extracted from TMDB, including cast names, director names, genres, and keywords. These features enable content-based similarity calculations.
- *Temporal Context:* The timestamp t associated with each interaction, providing insights into temporal patterns (time-of-day effects, preference evolution) and enabling recency-aware recommendations.

This formalization clarifies that our recommender system operates under significant uncertainty (partial observability) and must continuously adapt (dynamic environment) while optimizing multiple competing objectives (accuracy, diversity, coverage). These constraints fundamentally shape our algorithmic choices, as we will see in subsequent sections.

1.4 Recommender System Architectures

Despite the diversity of application domains, most recommender systems share a common three-stage pipeline that progressively narrows down the space of candidate items to produce final recommendations:

1. **Candidate Generation:** Given a target user, this stage rapidly filters the entire item catalog (potentially millions of items) down to a manageable set of hundreds or thousands of candidate items that are plausibly relevant. This stage prioritizes *recall*—we want to ensure that good recommendations are included in the candidate set, even at the cost of including some irrelevant items.

   Common techniques include:

   - Retrieving items similar to those the user has previously liked (content-based)
   - Identifying items popular among users with similar taste profiles (collaborative filtering)
   - Combining multiple retrieval strategies and taking their union

2. **Scoring:** Once candidates are generated, this stage assigns a precise numerical score to each item, typically representing the predicted rating or probability that the user will engage with the item. This stage uses more computationally expensive models (e.g., matrix factorization, neural networks) since it operates on a smaller set of candidates. The goal is to accurately rank candidates by predicted relevance, prioritizing *precision*.

3. **Re-ranking:** The final stage applies business logic, diversity constraints, and user-specific rules to reorder the scored candidates before presentation. This may involve:

    - Removing items the user has already consumed or explicitly disliked
    - Injecting diversity to avoid showing only similar items
    - Promoting fresh or trending content
    - Balancing personalization with exploration of new content

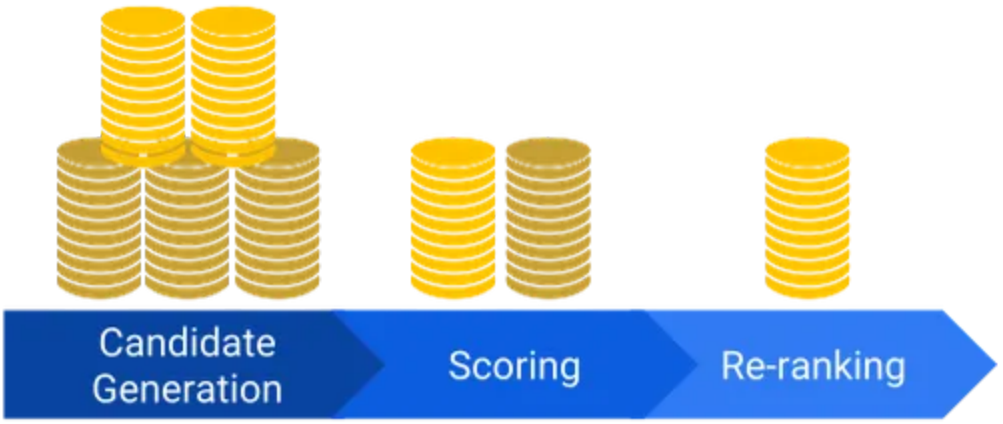


FIGURE 1.1: Three-stage recommender system pipeline: Candidate Generation narrows millions of items to thousands, Scoring ranks these candidates by relevance, and Re-ranking applies constraints to produce the final recommendation list.

This staged architecture allows systems to balance computational efficiency (candidate generation is fast but approximate) with prediction quality (scoring is expensive but accurate), while maintaining flexibility to incorporate business rules in the re-ranking stage.

1.5 Fundamental Approaches to Recommendation

Recommender systems employ three fundamental paradigms, each with distinct assumptions about how to identify relevant items. Understanding these paradigms is essential for selecting and combining techniques appropriately.

1.5.1 Content-Based Filtering

Content-based filtering recommends items by matching their characteristics to a user's preference profile. The core assumption is: *if a user liked certain items in the past, they will like similar items in the future.*

How It Works

The system maintains two types of profiles:

- **Item Profile:** A feature representation of each item derived from its intrinsic characteristics. For movies, this might include genre tags, cast members, director, keywords, and plot descriptions. These features are typically encoded as vectors in a high-dimensional space using techniques like TF-IDF (Term Frequency-Inverse Document Frequency).
- **User Profile:** A representation of the user's preferences, typically constructed by aggregating the item profiles of items the user has positively interacted with. For example, if a user has rated multiple Christopher Nolan films highly, the user profile will have high weight for "Christopher Nolan" and genres like "Thriller" and "Sci-Fi".

To generate recommendations, the system computes similarity between the user profile and candidate item profiles, ranking items by their similarity score. Items most similar to the user's historical preferences are recommended.

Advantages

- **Transparency:** Content-based systems provide interpretable explanations for recommendations. For example: "We recommend this movie because you enjoyed other films directed by Christopher Nolan" or "You've rated 3 action-thriller films highly."
- **No Cold-Start for New Items:** New items can be recommended immediately as long as their content features are available, without requiring any user interaction history.
- **User Independence:** Recommendations for one user don't depend on other users' data, avoiding privacy concerns and making the system robust to manipulation.

Limitations

- **Feature Engineering Dependency:** The quality of recommendations is fundamentally limited by the expressiveness of the item features. If important characteristics (e.g., "emotional tone" or "cinematography style") are not captured in the features, they cannot influence recommendations.
- **Overspecialization (Filter Bubble):** By always recommending items similar to past preferences, the system creates an echo chamber effect. Users are never exposed to content outside their established taste profile, limiting serendipitous discovery. A user who has only watched action movies will never receive recommendations for documentaries, even if they might enjoy them.
- **Limited Cross-Domain Knowledge:** Content-based systems cannot leverage insights across item types. The fact that users who enjoy "The Matrix" also tend to enjoy "Inception" (a pattern visible in collaborative data) cannot be discovered from content features alone.

1.5.2 Collaborative Filtering

Collaborative filtering takes a fundamentally different approach: it recommends items based on patterns in collective user behavior, without requiring knowledge

of item content. The core assumption is: *users who agreed in the past will agree in the future.*

For example, if users A and B both rated "The Shawshank Redemption" and "The Godfather" highly, and user A also loved "Pulp Fiction" while user B hasn't seen it, we can recommend "Pulp Fiction" to user B based on their similar taste profile with user A.

Memory-Based vs. Model-Based

Collaborative filtering can be implemented in two ways:

- **Memory-Based (Neighborhood Methods):** These methods directly compute similarities between users (user-based CF) or between items (item-based CF) using the rating matrix, then predict ratings based on weighted averages of similar users' ratings or similar items' ratings. These methods are intuitive but don't scale well to large datasets.
- **Model-Based (Latent Factor Models):** These methods, which we employ in this work, learn a compressed representation of the user-item interaction matrix by discovering latent factors that explain observed ratings. Techniques like Singular Value Decomposition++ (SVD++) and matrix factorization fall into this category. Model-based approaches scale better and often achieve superior predictive accuracy.

Model-Based Collaborative Filtering: Overview

Model-based collaborative filtering constructs a predictive model from historical user-item interactions to forecast future preferences. The workflow consists of three stages:

1. **Data Representation:** User-item interactions are organized into a rating matrix $R \in \mathbb{R}^{m \times n}$, where m is the number of users, n is the number of items, and R_{ui} represents user u's rating of item i. In practice, this matrix is extremely sparse—users typically rate less than 1% of available items.
2. **Model Training:** A matrix factorization technique (eg. SVD) decomposes the rating matrix into lower-dimensional latent factor matrices. These latent factors represent abstract user preferences (e.g., "preference for action

movies" or "preference for character-driven stories") and corresponding item characteristics, learned entirely from rating patterns without explicit feature engineering.

3. **Prediction:** For an unobserved user-item pair (u, i), the model predicts the rating by computing the dot product of the user's latent factor vector and the item's latent factor vector: $\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$, where $\mathbf{p}_u$ represents user u's latent preferences and $\mathbf{q}_i$ represents item i's latent characteristics.

Singular Value Decomposition++ (SVD++)

SVD++ is a matrix factorization technique that decomposes the user-item rating matrix into lower-dimensional latent factor representations. It learns vectors for users and items such that their dot product approximates the observed ratings, effectively capturing underlying preference patterns in a compact form. The model learns three sets of latent factors:

- **User Latent Factors** $P \in \mathbb{R}^{m \times k}$: Each row $\mathbf{p}_u$ represents user u's preferences across k latent dimensions.
- **Item Latent Factors** $Q \in \mathbb{R}^{n \times k}$: Each row $\mathbf{q}_i$ represents item i's characteristics across the same k dimensions.
- **Implicit Feedback Factors** $Y \in \mathbb{R}^{n \times k}$: Each row $\mathbf{y}_j$ captures implicit signals from item j. This allows the model to account for the fact that users who rate many horror movies (implicit signal) may have different preferences than users who rate few horror movies, even if their average ratings are similar.

where k is the number of latent dimensions, a hyperparameter typically set between 20-200 based on dataset size and complexity.

The predicted rating $\hat{r}_{ui}$ combines the user's explicit latent preferences with implicit feedback from their interaction history:

$$\hat{r}_{ui} = \mu + b_u + b_i + \mathbf{q}_i^T \left(\mathbf{p}_u + |I_u|^{-1/2} \sum_{j \in I_u} \mathbf{y}_j \right) \tag{1.1}$$

where:

- μ is the global average rating across all users and items
- b_u is user u's rating bias (some users consistently rate higher than others)
- b_i is item i's rating bias (some items are universally more popular)
- I_u is the set of items rated by user u
- The term $|I_u|^{-1/2} \sum_{j \in I_u} \mathbf{y}_j$ represents the normalized implicit feedback signal from all items the user has interacted with

This formulation enables SVD++ to leverage the fact that *which items* a user has rated provides information beyond *how* they rated those items, leading to improved predictions, especially for users with sparse rating histories.

Advantages of Collaborative Filtering

- **No Content Knowledge Required:** The system discovers patterns purely from interaction data, making it domain-agnostic and eliminating the need for expensive feature engineering.
- **Serendipity:** Can recommend items that are dissimilar in content but appeal to similar users, enabling surprising and novel recommendations that break filter bubbles.
- **Quality Signal:** Leverages the "wisdom of the crowd"—popular items among similar users are likely to be genuinely good recommendations.

Limitations of Collaborative Filtering

- **Cold-Start Problem:** New users with no rating history cannot receive personalized recommendations, and new items with no ratings cannot be recommended to anyone. This creates a chicken-and-egg problem for onboarding.
- **Data Sparsity:** The user-item rating matrix is typically 99%+ sparse. Finding users with overlapping rated items becomes increasingly difficult, reducing prediction confidence.
- **Popularity Bias:** Collaborative methods tend to over-recommend popular items (which have more rating data) at the expense of niche content, exacerbating the "rich get richer" effect.

1.5.3 Hybrid Recommender Systems

Hybrid systems combine content-based and collaborative filtering approaches to leverage the strengths of each while mitigating their individual weaknesses. Common hybridization strategies include:

- **Weighted Hybrid:** Combine the scores from both methods using a learned weighting: $\hat{r}_{ui} = \alpha \cdot \text{score}_{\text{content}} + (1 - \alpha) \cdot \text{score}_{\text{collaborative}}$
- **Switching Hybrid:** Use content-based filtering for cold-start cases (new users/items) and collaborative filtering for established users/items with sufficient interaction history.
- **Feature Augmentation:** Use content features as additional inputs to collaborative filtering models, or vice versa.

By combining approaches, hybrid systems can provide accurate, diverse recommendations while maintaining robustness to cold-start scenarios and data sparsity—an approach we explore in our implementation.

1.6 Similarity Measures: Quantifying Proximity

Both content-based and collaborative filtering require a notion of *similarity*—a way to quantify how "close" two items or two users are in some feature space. This similarity metric is fundamental to recommendation: the system's ability to suggest relevant items depends entirely on its ability to measure relevance through similarity.

The choice of similarity measure depends on how items or users are represented. For vector representations (as in TF-IDF or latent factor models), cosine similarity is the most widely used metric due to its invariance to vector magnitude and intuitive geometric interpretation.

1.6.1 Cosine Similarity

Cosine similarity measures the cosine of the angle between two vectors in a multi-dimensional space. Unlike Euclidean distance, which measures the magnitude of

the difference between vectors, cosine similarity focuses on their orientation—two vectors pointing in the same direction are similar, regardless of their length.

This property is crucial for recommendation systems: a user who has rated 10 movies and a user who has rated 1,000 movies might have similar *taste* (same orientation in preference space) even though their rating vectors have very different magnitudes.

Mathematical Derivation

Starting from the Euclidean dot product formula:

$$\langle \mathbf{a}, \mathbf{b} \rangle = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos \theta \tag{1.2}$$

where $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the inner product[1], $\|\mathbf{a}\|$ and $\|\mathbf{b}\|$ are the Euclidean norms (lengths) of the vectors, and θ is the angle between them.

Rearranging to solve for $\cos\theta$:

$$\cos \theta = \frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\|\mathbf{a}\| \, \|\mathbf{b}\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \sqrt{\sum_{i=1}^{n} b_i^2}} \tag{1.3}$$

This formula computes the cosine similarity between vectors $\mathbf{a}$ and $\mathbf{b}$.

Interpretation

The cosine similarity value ranges from -1 to 1:

- $\cos\theta = 1$: The vectors are perfectly aligned (parallel), indicating maximum similarity
- $\cos\theta = 0$: The vectors are orthogonal (perpendicular), indicating no relationship
- $\cos\theta = -1$: The vectors point in opposite directions, indicating maximum dissimilarity

In recommender systems working with non-negative features (e.g., TF-IDF weights or rating counts), the cosine similarity is bounded in $[0, 1]$, providing a natural

similarity score where 1 represents identical items and 0 represents completely unrelated items.

Why Cosine Similarity Matters for Recommendations

Consider a movie recommendation scenario. Movie A has the feature vector $[0.5, 0.3, 0.2, 0.0]$ (representing weights for genres: action, comedy, drama, horror). Movie B has the vector $[0.5, 0.3, 0.2, 0.0]$ (identical), and Movie C has $[1.0, 0.6, 0.4, 0.0]$ (proportionally identical but with larger values, perhaps due to having more metadata).

Using Euclidean distance, Movie A would appear "closer" to Movie B than to Movie C. However, cosine similarity correctly identifies that Movies A and C have identical taste profiles—they're just represented at different scales. This scale-invariance makes cosine similarity ideal for comparing items with varying amounts of metadata or users with different numbers of ratings.

1.7 Challenges in Recommender Systems

Despite their widespread success, recommender systems face several fundamental challenges that constrain their effectiveness and require careful algorithmic design to mitigate.

1.7.1 The Cold-Start Problem

The cold-start problem arises when the system lacks sufficient information to make reliable predictions. This manifests in two forms:

New User Cold-Start: When a user first joins the platform, they have no rating history. Collaborative filtering cannot function without observing the user's preferences, forcing the system to either ask the user to rate items explicitly (adding friction to onboarding) or fall back to non-personalized recommendations (popular items, trending content).

New Item Cold-Start: Newly added items have no ratings, making them invisible to collaborative filtering. This creates a vicious cycle: new items cannot be recommended because they have no ratings, but they cannot acquire ratings

if they're never recommended. Content-based filtering partially addresses this by recommending new items based on their features, but cannot leverage quality signals from the crowd.

Mitigation Strategies: Hybrid systems that combine content-based features with collaborative signals can provide reasonable recommendations even in cold-start scenarios. Additionally, exploration strategies (e.g., epsilon-greedy recommendation) can deliberately surface new items to gather initial feedback.

1.7.2 Data Sparsity

In typical recommender systems, users rate less than 1% of available items, resulting in an extremely sparse user-item rating matrix. This sparsity has several consequences:

- **Difficulty Finding Similar Users:** In collaborative filtering, computing user similarity requires finding users who have rated overlapping items. With high sparsity, these overlaps become rare, reducing the reliability of similarity estimates.
- **Unreliable Item Statistics:** Items with very few ratings have unreliable average ratings—a single 5-star rating gives a misleading quality signal.
- **Reduced Model Accuracy:** Matrix factorization models must extrapolate from limited observed data, increasing prediction uncertainty.

Mitigation Strategies: Dimensionality reduction techniques like SVD++ help by discovering latent patterns that generalize across sparse observations. Regularization prevents overfitting to the limited available data. Incorporating implicit feedback (views, clicks, time spent) adds density to the interaction matrix.

1.7.3 Scalability

Modern platforms must serve millions of users and catalog millions of items, requiring real-time recommendations with millisecond latency. This creates computational challenges:

- **Matrix Factorization Complexity:** Training models on massive user-item matrices is computationally expensive, requiring distributed computing infrastructure.
- **Real-Time Scoring:** Scoring millions of candidate items for every user request is infeasible. This necessitates the candidate generation → scoring → re-ranking pipeline described in Section 1.4, where inexpensive methods quickly narrow down candidates before applying expensive models.
- **Model Update Frequency:** User preferences and item catalogs change continuously. The system must balance model freshness (frequent retraining) with computational cost.

Mitigation Strategies: Approximate nearest neighbor algorithms enable fast candidate retrieval. Distributed matrix factorization frameworks (e.g., Spark MLlib) parallelize computation. Incremental learning algorithms update models without full retraining.

These challenges motivate our hybrid approach, combining the complementary strengths of content-based and collaborative filtering while employing efficient candidate generation and scoring strategies.

Chapter 2

Problem Definition

2.1 Problem Statement

Within the *Business Understanding* phase, the problem is defined by translating an application need into a precise analytical task. Movie streaming platforms face the challenge of helping users discover relevant content from catalogs containing thousands of titles. Manual browsing is inefficient, and users often struggle to find movies matching their preferences. This work considers the development of a recommendation system that, given historical evidence of user preferences and interactions, produces personalized movie suggestions over a catalogue of items.

Let $\mathcal{U}$ denote the set of users and $\mathcal{I}$ the set of items (movies). The available input data are modeled as a set of logged interactions

$$\mathcal{D} = \{(u, i, r_{ui}, t)\}, \tag{2.1}$$

where $u \in \mathcal{U}$ is a user identifier, $i \in \mathcal{I}$ is an item identifier, $r_{ui} \in \mathbb{R}$ is an observed rating (e.g., on a scale from 1 to 5), and t is a timestamp indicating when the interaction occurred.

To enable offline evaluation, the dataset is partitioned into disjoint training and test sets based on temporal ordering:

$$\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}, \quad \mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset, \tag{2.2}$$

where all interactions in $\mathcal{D}_{\text{train}}$ occur before those in $\mathcal{D}_{\text{test}}$ according to their timestamps.

The learning task is to estimate a scoring function $\hat{r} : \mathcal{U} \times \mathcal{I} \to \mathbb{R}$ from the training data that assigns a predicted relevance score $\hat{r}(u, i)$ to each candidate item for a given user. The function $\hat{r}$ is learned by minimizing a prediction error over the training set. For explicit rating prediction, a common objective is to minimize the regularized mean squared error:

$$\min_{\hat{r}} \sum_{(u,i,r_{ui},t)\in\mathcal{D}_{\text{train}}} (r_{ui} - \hat{r}(u, i))^2 + \lambda\Omega(\hat{r}), \tag{2.3}$$

where $\Omega(\hat{r})$ is a regularization term (e.g., L2 penalty on model parameters) and $\lambda > 0$ is a regularization coefficient.

For a target user u, let $\mathcal{I}_u^{\text{train}}$ denote the set of items for which the user has logged interactions in the training set:

$$\mathcal{I}_u^{\text{train}} = \{i \in \mathcal{I} : \exists(u, i, r_{ui}, t) \in \mathcal{D}_{\text{train}}\}. \tag{2.4}$$

The expected system output is a top-K ranked list of items not in the user's training history, namely

$$\pi_u^{(K)} = \text{top-}K \text{ items from } \{i \in \mathcal{I} \setminus \mathcal{I}_u^{\text{train}}\} \text{ ranked by } \hat{r}(u, i) \text{ in descending order,} \tag{2.5}$$

which can be written more formally as selecting the set $S_u \subseteq \mathcal{I} \setminus \mathcal{I}_u^{\text{train}}$ with $|S_u| = K$ such that

$$\min_{i\in S_u} \hat{r}(u, i) \geq \max_{j\in(\mathcal{I}\setminus\mathcal{I}_u^{\text{train}})\setminus S_u} \hat{r}(u, j), \tag{2.6}$$

and ordering items in S_u by decreasing score.

The scope of the study is restricted to offline learning and offline evaluation using historical data; interactive recommendation in a live environment is not considered in the present formulation.

2.2 Requirements Analysis

The problem definition implies a set of requirements that the system must satisfy.

Functional requirements include:

- producing a ranked recommendation list $\pi_u^{(K)}$ for each target user based on the available interaction history;

- supporting batch generation of recommendations for a set of users to enable offline evaluation and analysis;
- enabling reproducible dataset partitioning and evaluation under a clearly specified protocol.

Non-functional requirements include:

- *scalability* with respect to the number of users, items, and interactions, so that the evaluation protocol remains feasible on realistic dataset sizes;
- *reproducibility* of data processing and evaluation, including deterministic configuration of data splits and reporting of all hyperparameters relevant to the experimental protocol;
- *interpretability of reporting*, in the sense that the produced artifacts (data summaries, evaluation outputs) support transparent scientific communication.

2.3 Business Criteria and Success Objectives

At the application level, the system is intended to support user decision-making by surfacing movies that are likely to be relevant to individual preferences. Accordingly, success objectives are defined in terms of measurable offline criteria that approximate utility under the available logs.

The **primary success objective** is the production of accurate personalized rankings, assessed using standard top-K ranking metrics computed on held-out interactions in $\mathcal{D}_{\text{test}}$. These metrics include precision at K (P@K), recall at K (R@K), mean average precision (MAP), and normalized discounted cumulative gain (nDCG@K).[1] The **secondary objectives** include maintaining catalogue coverage and providing stable performance across user segments with different activity levels.

The evaluation objectives are aligned with the intended usage scenario: given a user context represented by historical interactions in $\mathcal{D}_{\text{train}}$, the system should prioritize items that the user is likely to engage with in subsequent interactions captured in $\mathcal{D}_{\text{test}}$ under the adopted feedback definition.

2.4 Constraints and Assumptions

The formulation assumes that historical interaction logs are available and that user and item identifiers are consistent within the selected snapshot. It is assumed that timestamps reflect the chronological order of interactions at the granularity required by the evaluation protocol, enabling a meaningful temporal train-test split that mimics real-world deployment conditions where the system must predict future interactions.

From a data perspective, it is assumed that the observed rating r_{ui} is a valid proxy for user preference under the chosen task definition. From a computational perspective, it is assumed that the system can generate rankings for the evaluation user set within the available compute budget, and that the chosen protocol does not require exhaustive scoring of all user–item pairs when infeasible. In practice, efficient candidate generation or approximate nearest neighbor methods may be employed to make ranking tractable at scale.

For movie recommendation specifically, the following domain assumptions apply:

- User preferences are relatively stable over the time scale captured by the dataset, though temporal dynamics may be present;
- The rating scale is consistent across users and reflects genuine preference signals rather than pure noise;
- The cold-start problem (new users or items with no interaction history) is acknowledged but not the primary focus of this formulation.

2.5 Technology and Tool Selection

The experiments are carried out in the cloud using the free version of Google Colaboratory notebooks with Python as the implementation language. Google Colab provides virtual machines with 12GB of RAM and Tesla K80 Graphical Processing Units (GPU) or Tensor Processing Units (TPU) as hardware accelerators depending on the availability of those machines. This computational environment constrains the scale of datasets and models that can be feasibly evaluated, but provides sufficient resources for standard benchmark datasets in the movie recommendation domain (e.g., MovieLens variants).

Chapter 3

Dataset Description and Data Understanding

In this section, we present a general exploration of The Movies Dataset, combining data from MovieLens and the TMDb API. These datasets contain movie metadata and user-item interactions and served as the **foundation** for carrying out experiments on various families of recommendation systems.

3.1 Data Sources

MovieLens is a web-based recommender system from GroupLens, a research lab at the University of Minnesota. It recommends movies to users based on their film preferences, utilizing collaborative filtering of members' movie ratings and reviews.

The Movie Database (TMDb) is a community-built movie and TV database used by developers and enthusiasts worldwide. It provides detailed information, posters, trailers, and data integration via a powerful API.

The dataset used in this work is an ensemble of data collected from both sources:

- Movie Details and Credits are gathered from the TMDb Open API.
- Movie Links and Ratings are obtained from the official GroupLens website.
- Additional reviews information gathered form the Rotten Tomatoes movie reviews dataset, obtained via Kaggle.

The MovieLens dataset is distributed in multiple comma-separated values (CSV) files:

- `movies.csv`: Contains basic metadata (ID, title, genres).
- `ratings.csv`: Stores explicit user ratings, user IDs, movie IDs, and timestamps.
- `tags.csv`: Includes user-generated tags describing movies.
- `links.csv`: Maps MovieLens IDs to external identifiers (IMDb, TMDb).

The Rotten tomatoes dataset is distributed in the following manner:

- **id**: Unique identifier for each review.
- **stringlengths**: Attribute related to string lengths (details unspecified, possibly for analysis or filtering purposes).
- **reviewId**: Unique identifier for each review (integer).
- **creationDate**: Date when the review was created.
- **criticName**: Name of the critic providing the review.
- **isTopCritic**: Boolean flag indicating if the critic is recognized as a top critic.
- **originalScore**: The original rating score given by the critic (may include null values).
- **reviewState**: Review classification state (e.g., "fresh" or "rotten").
- **publicatioName**: Name of the publication where the review was published.
- **reviewText**: Full text of the review.
- **scoreSentiment**: Sentiment associated with the review score (e.g., "POSITIVE" or "NEGATIVE").
- **reviewUrl**: URL linking to the full review.

3.1.1 Movies Metadata

The Movies Dataset contains metadata for 85,235 movies featured in the `tmdb_movies.parquet` file. Each movie has 13 attributes. The table below lists selected attributes with their types and descriptions. The actual number of movies used was reduced to 12,468 due to missing Movie id's and genre attributes.

TABLE 3.1: Description of selected attributes from the Movies Metadata dataset

Attribute	**Object Type**	**Non-nulls**	**Description**
budget	string	45,466	Budget for the movie production in USD
genres	stringified JSON	45,466	Identifier and name of the movie genres
tmdb_id	string	45,466	TMDb identifier
original_language	string	45,455	Original language of the movie
original_title	string	45,466	Title in original language
popularity	float	45,461	Popularity metric provided by TMDb

Movies can belong to 20 distinct genres. As shown in Figure 3.1, most movies belong to 1 to 3 genres. Not all genres are equally represented: Drama is the most prevalent (33,500 records), while Westerns and TV Movies have fewer than 1,250 records.

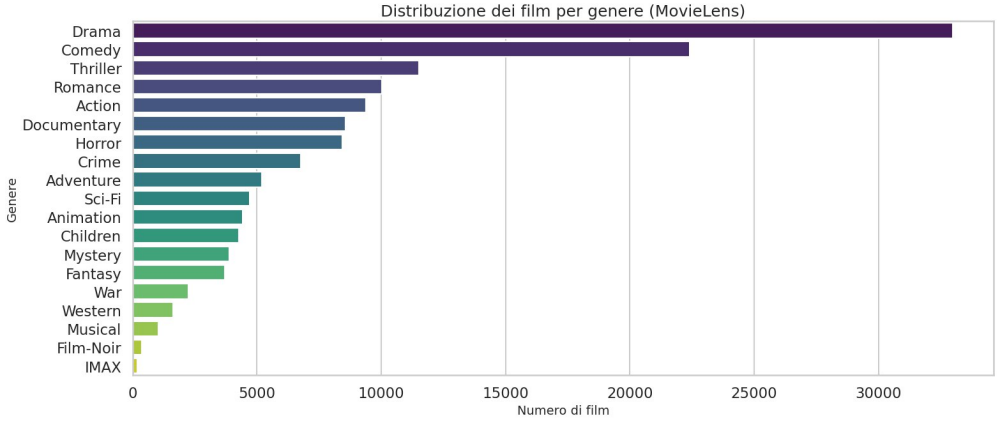


FIGURE 3.1: Film distribution based on genre.

Release Date: Each movie has a release date ranging from 1874 to 2023 (format "yyyy-mm-dd"). As depicted below, the dataset spans from the late 19th century to 2023, with the majority of films released between 2000 and 2010.

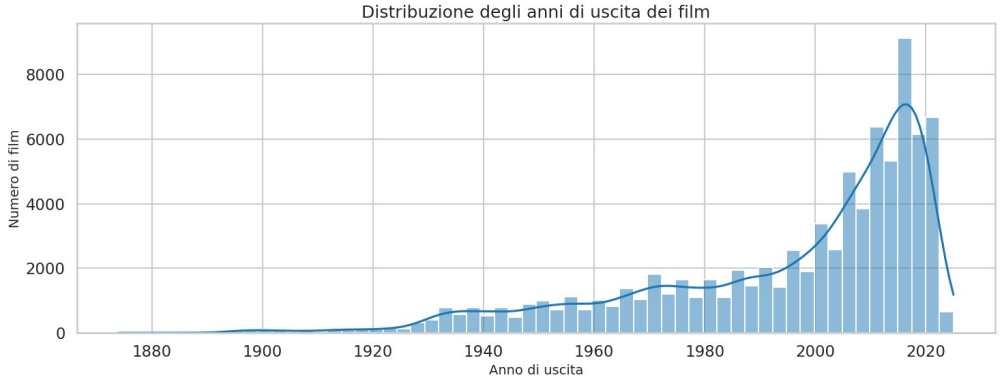


FIGURE 3.2: Film release year distribution.

3.1.2 Cast and Crew Credits Metadata

Credits information was gathered from the TMDb Open API and stored in `tmdb_credits.parquet`. Unlike traditional JSON arrays, this file is denormalized: each row represents a single credit entry (cast or crew).

The file contains approximately 3.8 million rows. For the 85,235 movies in our processed dataset, cast information is available for 82,025 movies (96.27% coverage).

TABLE 3.2: Description of attributes in the Credits dataset

Attribute	**Type**	**Description**
tmdb_id	integer	TMDb identifier of the movie
type	string	"cast" or "crew"
person_id	integer	Unique TMDb identifier for the person
name	string	Full name of the person
character	string	Name of the character (null for crew)
order	float	Billing order (null for crew)
job	string	Specific role (null for cast)
department	string	Department (null for cast)

Focusing on cast entries, we observe:

- 1,701,269 total cast appearances (roles).
- 623,056 unique actors.
- Average cast size: 20.74 members per movie (Median: 16).

Actor appearances follow a long-tail distribution. The top 10 most frequent actors are shown in Figure 3.4.

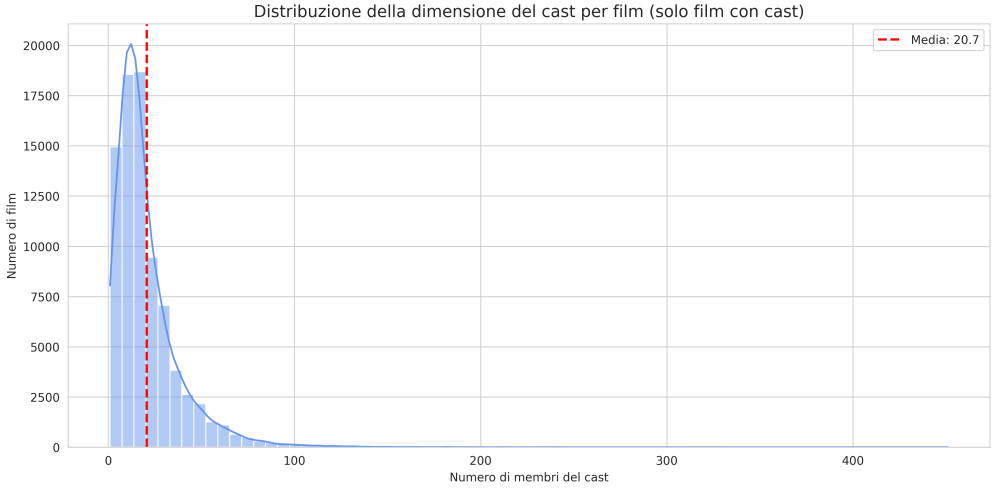


FIGURE 3.3: Distribution of cast size per film.

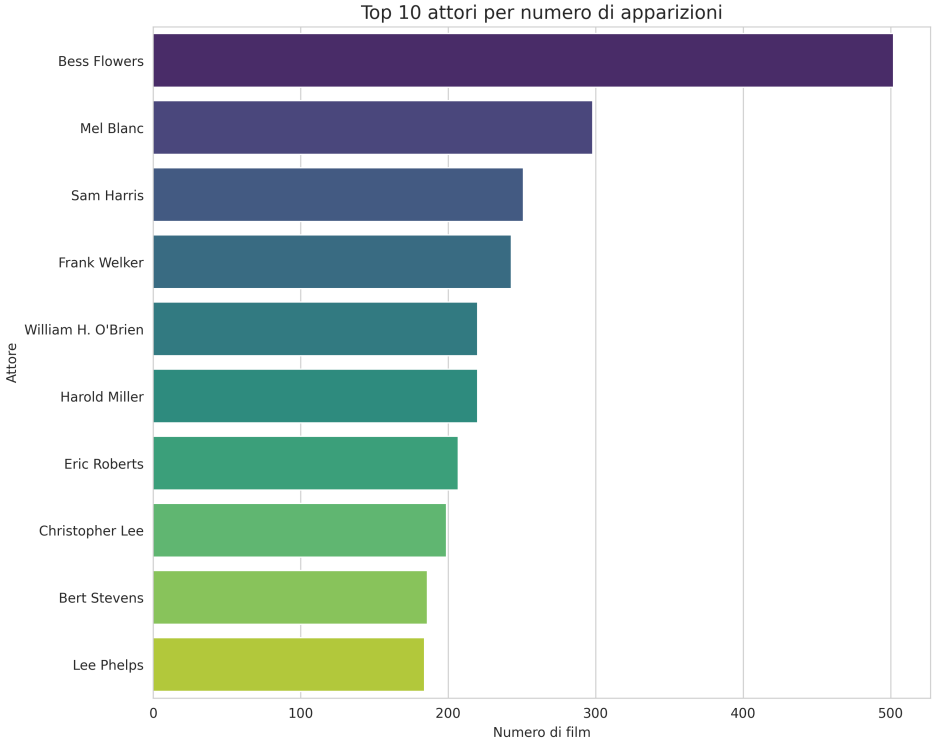


FIGURE 3.4: Top 10 actors by number of appearances.

3.1.3 Critic Reviews Metadata

The Rotten Tomatoes Movie Reviews dataset includes 1.44 million reviews. Attributes are listed below:

TABLE 3.3: Description of attributes from Rotten Tomatoes Reviews

Attribute	Type	Non-nulls	Description
id	string	1,444,963	Unique identifier
reviewId	integer	1,444,963	Numeric identifier
creationDate	datetime	1,444,963	Date of publication
criticName	string	1,444,963	Critic's name
isTopCritic	boolean	1,444,963	Top critic indicator
originalScore	string	1,009,745	Original rating score
reviewState	string	1,444,963	Publication status
publicationName	string	1,444,963	Publisher name
reviewText	string	1,375,738	Text content
scoreSentiment	string	1,444,963	Sentiment label
reviewUrl	string	1,234,038	URL to original review

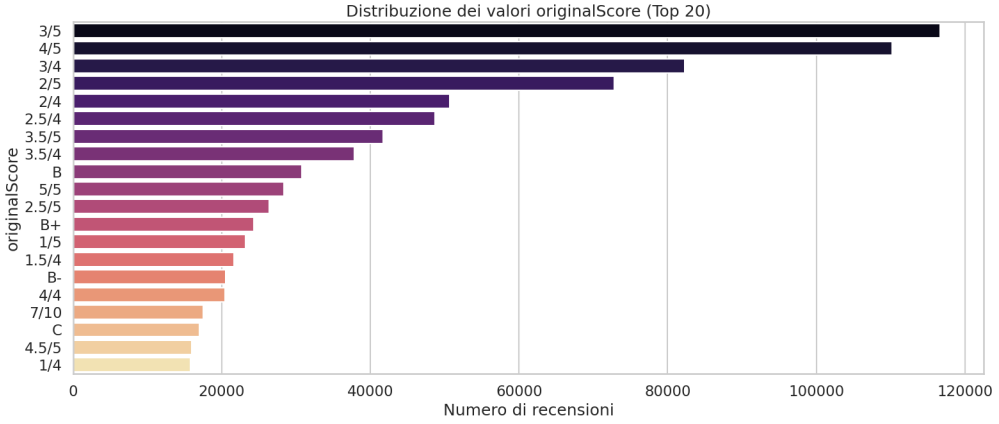


FIGURE 3.5: Distribution of the originalScore values from the Rotten Tomatoes Dataset.

3.2 Data Collection and Sampling Strategy

To ensure reproducibility and manage computational constraints, we implemented a deterministic pipeline to transform the raw data into an analysis-ready corpus.

- **Inclusion criteria (ID Resolution):** We retained only those interactions where the MovieLens identifier (`movieId`) could be successfully mapped to a valid TMDb identifier (`tmdb_id`). This ensured that every user rating in the system corresponded to a movie with rich metadata (Cast, Director, Genres) available in our target analysis set.

- **Memory Management Strategy:** The original dataset contained approximately **33 million interactions**. Preliminary analysis indicated that loading this full history would require over **3.00 GB** of RAM for the raw data

structures alone, creating significant bottlenecks during the complex matrix factorization steps. To ensure stability within the Google Colab environment, we set a strict target cap of **2,208,239** ratings.

- **Sampling Design (Stratified Downsampling):** Unlike random subsampling, which risks eliminating users entirely, we implemented a **stratified clipping strategy** to preserve the user diversity while reducing data volume.

  1. We calculated a dynamic quota of ratings per user: $Q = \lfloor \frac{\text{Target Size}}{\text{Unique Users}} \rfloor$.
  2. For each user u, we retained a maximum of Q ratings (specifically, $\min(|R_u|, Q)$).

  This approach effectively "clipped" the long tail of power users—who often introduce bias—while preserving the complete history of standard users.

- **Final Corpus Size:** The resulting dataset contains exactly **2,208,239** ratings. This subset ensures a manageable memory footprint while maintaining the statistical properties required for training the hybrid models.

3.3 Data Structure and Feature Description

The final dataset is represented as a set of relational tables:

- **Interactions table** $\mathcal{D}_I$: tuples (u, i, y, t) where u is a user, i an item, y the feedback signal, and t the timestamp.
- **Items table** $\mathcal{D}_X$: item-level descriptors (metadata, embeddings).
- **Users table** $\mathcal{D}_U$: user attributes.

Table 3.4: Feature summary for the final dataset.

Feature	Type	Description
user_id	categorical	Unique user identifier (anonymized)
item_id	categorical	Unique item identifier
feedback_y	numeric	Rating value (0.5 - 5.0)
timestamp_t	datetime	Interaction time
item_genres	list	Genre tags associated with the movie
release_year	numeric	Year of movie release

3.4 Data Quality Assessment

We assessed the impact of our sampling strategy by comparing the statistical properties of the original merged dataset against the final processed corpus used for training. The quantitative comparison is presented in Table 3.5.

Table 3.5: Comparison between Original and Final Processed Dataset statistics.

Statistic	**Original Data**	**Final Processed Data**
Number of Users ($\|\mathcal{U}\|$)	330,975	10,000
Number of Items ($\|\mathcal{I}\|$)	83,239	12,468
Number of Ratings ($\|\mathcal{D}_I\|$)	33,832,162	2,208,239
Matrix Density	0.1228%	0.1094%[1]
Sparsity	99.8772%	99.8906%[2]

Interpretation of Statistics

As evidenced by the table, the sampling process reduced the volume of interactions by approximately 93.47%, resulting in a final corpus of **2,208,239 interactions**. This reduction was necessary to strictly adhere to the 3GB RAM constraints of the experimental environment.

It is worth noting that the **Matrix Density** substantially increased (from 0.1228% to 1.771%). This increase is a direct consequence of the *stratified clipping* strategy described in Section 3.2. By selecting a balanced subset of users across activity levels and limiting the maximum number of ratings per user, we effectively removed the sparsest parts of the matrix (inactive users) while capping the influence of power users. This strengthens the overall signal for model training, enhances the reliability of similarity estimates, and reduces popularity bias by preventing overfitting to the preferences of a small minority of hyper-active users.

The final sparsity of **99.89%** entails specific challenges for the recommender system, particularly regarding **Diversity**[3]. To overcome the lack of explicit signal in such a sparse matrix, our methodology employs dimensionality reduction

[1]Matrix Density is defined as the ratio of observed ratings to the total possible user-item pairs: $Density = \frac{|\mathcal{D}_I|}{|\mathcal{U}|\times|\mathcal{I}|}$. It represents the "signal strength" available to the model.

[2]Sparsity is the complement of density ($1 - Density$). A sparsity of 99.89% implies that 99.89% of the entries in the user-item matrix are empty (unknown), posing a challenge known as the "Cold Start" problem.

(SVD++) and hybrid techniques to infer latent preferences where direct overlaps are missing.

3.5 Identified Challenges and Limitations

We identified several challenges inherent to the dataset:

- **Sparsity:** The interaction matrix is highly sparse ($\approx$ [**99.8906**]%), requiring robust collaborative filtering techniques.
- **Cold-start:** New items lack interaction history.
- **Selection bias:** Observed ratings are not missing-at-random; users tend to rate items they already expect to like.

These limitations motivate the use of hybrid recommendation strategies and popularity debiasing in our subsequent experiments.

Chapter 4

Data Preprocessing and Engineering

In this chapter, we detail the transformation of raw data into a structured format suitable for modeling. Our preprocessing pipeline addresses three fundamental challenges: (1) integrating heterogeneous data sources with different identifiers and formats, (2) extracting meaningful signals from unstructured text and metadata, and (3) engineering features that capture both user behavior and item characteristics.

The pipeline consists of three main stages: **Cleaning and Integration** (handling missing values and merging external sources), **Feature Engineering** (deriving signals from timestamps and text), and **Feature Extraction** (vectorizing unstructured metadata). Finally, we present a comprehensive **Exploratory Data Analysis (EDA)** to validate the dataset properties and uncover patterns that will inform our modeling choices.

4.1 Data Cleaning and Integration

The first stage involved ensuring referential integrity between the interaction data (MovieLens), content metadata (TMDb), and auxiliary reviews (Rotten Tomatoes). Our goal was to create a unified dataset where every rating could be enriched with both structured metadata (actors, directors, genres) and external expert opinions (critic reviews).

4.1.1 Structural Alignment and Filtering

To prepare the data for content-based filtering, we needed to ensure that every movie in our ratings dataset had corresponding metadata. This required careful alignment between our data sources.

We implemented a strict filtering pipeline to ensure a 1:1 consistency between items and their features:

- **Dataset Intersection:** We performed an inner join between the movie catalog and the credits database. This step eliminated "orphan" records—entries in the credits file referring to movies not present in our target analysis set. Without this alignment, our content-based models would fail when attempting to retrieve features for movies that existed in the ratings but not in the metadata.
- **Granular Cast Filtering:** Unlike standard datasets that treat the cast as a single text block, we utilized the `order` attribute available in the credits file to apply a relevance filter. We retained only the **top-5 billed actors** (where `order` < 4, with order positions starting from 0) for each movie.

  This threshold was chosen based on the observation that the top-billed actors—typically the protagonist, antagonist, and primary supporting roles—are the most recognizablefluential in defining a movie's identity and appeal.. Actors appearing fifth or lower in the billing order are often less famous and play smaller roles that contribute less to a viewer's decision to watch a film. By filtering beyond position 4, we significantly reduced the dimensionality of the feature space while filtering out extras and minor roles that act as noise in similarity calculations. This enables our content-based model to focus on the cast members most likely to influence user preferences.

4.1.2 Critic Reviews Processing (Rotten Tomatoes)

The Rotten Tomatoes Movie Reviews dataset initially comprised 1,444,963 reviews. Unlike the structured integer ratings in MovieLens (1-5 stars), this dataset presented two major integration challenges: unstructured rating formats and non-standard movie identifiers. Successfully integrating this data would enrich our system with expert critical opinions, providing a valuable signal beyond user-generated ratings.

The dataset is characterized by heterogeneity in both rating scales and movie identification:

TABLE 4.1: Description of attributes from Rotten Tomatoes Reviews

Attribute	Type	Non-nulls	Description
id	string	1,444,963	Alphanumeric slug (e.g., `the_matrix_1999`)
originalScore	string	1,009,745	Raw score (e.g., "A-", "4/5", "80%")
reviewText	string	1,375,738	Full text content of the review

Integrating this data required a two-step engineering pipeline to address both the rating format heterogeneity and the identifier mismatch:

4.1.2.1 Score Normalization and VADER Imputation

The `originalScore` attribute presented high variance in format (fractions like "3.5/4", percentages like "80%", letter grades like "B+"). We normalized these to a standard interval $y \in [0, 5]$ to match our MovieLens rating scale.

However, **444,174 reviews** (approximately 30% of the dataset) contained review text but lacked a numerical score. Simply discarding these reviews would waste valuable information. To utilize this data, we employed Sentiment Analysis using the **VADER (Valence Aware Dictionary and sEntiment Reasoner)** lexicon—a rule-based sentiment analyzer specifically designed for social media and informal text, making it well-suited for movie reviews.

VADER produces a *compound* sentiment score $C \in [-1, 1]$, where -1 represents extremely negative sentiment and +1 represents extremely positive sentiment. We projected this onto our target rating scale using the linear transformation:

$$\hat{y} = \frac{C + 1}{2} \times 5$$

This linear mapping ensures that neutral sentiment ($C = 0$) maps to the midpoint of our scale (2.5 stars), while extremely positive sentiment ($C = 1$) maps to 5 stars and extremely negative ($C = -1$) maps to 0 stars. While this imputation introduces some noise compared to explicit scores, it allowed us to recover nearly half a million interactions that would otherwise have been discarded, significantly enriching our dataset.

4.1.2.2 Entity Resolution (ID Mapping)

The Rotten Tomatoes dataset identifies movies using URL slugs (e.g., `mission_impossible_1996`) rather than the integer TMDb IDs used in our primary dataset. To link these external reviews to our MovieLens-TMDb foundation, we developed a deterministic resolution pipeline querying the TMDb API.

Our approach employed a rule-based matching strategy: we parsed each slug to extract the candidate title and release year, queried the TMDb Search API, and accepted matches where both the title similarity was high and the release year aligned exactly. This deterministic approach ensured reproducibility and allowed us to manually validate ambiguous cases.

The process successfully mapped **60,923 unique movies** (an 87.9% hit rate from 69,263 unique slugs), adding over **1.3 million expert reviews** to our knowledge base. This enrichment provides a dense layer of professional critical opinion to complement user-generated ratings, enabling hybrid models that can leverage both signals.

4.2 Feature Engineering

With our data sources integrated, we performed comprehensive feature engineering to extract behavioral signals and create model-ready representations. This section describes how we transformed raw timestamps, text metadata, and interaction histories into features that capture user preferences, item characteristics, and temporal patterns.

4.2.1 Textual Preprocessing (Metadata Soup)

For content-based recommendation, we need to represent each movie as a combination of its attributes (genres, cast, director). However, these attributes exist as separate structured fields. Our challenge was to create a unified text representation that could be fed into a vectorization algorithm while preserving the semantic meaning of each entity.

We call this unified representation a "metadata soup",a concatenated string containing all relevant movie attributes. To maximize the signal-to-noise ratio for the

Content-Based model, we applied specific text normalization rules to the extracted entities (Actors, Directors, and Genres):

- **Entity Tokenization:** We converted alTom Hanks lowercase and removed whitespace (e.g., transforming "Matt Damon" " into "tomhanks"). This operation is critical for the TF-IDF Vectorizer to function correctly. Without this unification, the algorithm would treat the first name "Tom" as a separate, high-frequency token appearing in thousands of documents (Tom Hanks, Tom Cruise, Tom Holland). The vectorizer would then heavily penalize this common token, losing the specific identity of "Tom Hanks" as a unique entity. By merging first and last names, we created unique tokens representing specific entities that can be properly weighted by their discriminative power.
- **Metadata Soup Construction:** The processed features were concatenated into a single "metadata soup" string for each movie. For example, a movie might have the metadata soup: "actionadventure christophernolan christianbale michaelcaine". This composite field serves as the input document for vectorization, allowing the model to compute similarity based on a holistic view of the movie's attributes rather than treating each attribute independently.

This preprocessing ensures that our content-based model can identify movies with shared directors, overlapping cast members, or similar genre combinations through the TF-IDF similarity mechanism described in Section 4.3.

4.2.2 Temporal and Aggregate Features

Beyond content attributes, user behavior and item popularity provide crucial signals for collaborative filtering. We derived several numerical features to capture these patterns.

Temporal Decomposition

The Unix timestamp associated with each rating was converted to datetime and decomposed into interpretable temporal features: `year`, `month`, `day_of_week`, and `hour`. These features enable us to detect potential temporal patterns, such as

whether users rate movies differently on weekends versus weekdays, or whether rating behavior has evolved over time.

User-Level Aggregates

For each user, we calculated the following statistics across their entire rating history:

- `avg_rating`: The user's mean rating across all movies. This captures user-specific rating bias—some users consistently rate higher than others.
- `rating_std`: The standard deviation of the user's ratings, indicating rating consistency. Low values suggest the user rates most movies similarly (less discriminating), while high values indicate strong preferences (more critical).
- `num_ratings`: Total number of ratings provided. This serves as a proxy for user engagement and the reliability of their rating patterns.

Movie-Level Aggregates

For each movie, we calculated analogous statistics:

- `movie_avg_rating`: The movie's mean rating across all users. This represents the consensus quality assessment.
- `movie_rating_std`: The standard deviation of ratings received, indicating controversiality. High values suggest polarizing movies that some users love and others hate.
- `movie_num_ratings`: Total number of ratings received, serving as a popularity metric.

These aggregated features were merged back into the rating-level table, creating an enriched dataset suitable for collaborative filtering. As we will see in Section 4.4, these features reveal strong patterns—for example, individual ratings are highly correlated with both the user's historical average and the movie's consensus rating, suggesting that both user bias and item quality are major drivers of observed ratings.

4.3 Text Feature Extraction: TF-IDF

To enable content-based filtering, we must transform the unstructured "metadata soup" strings into a structured vector space where we can compute similarity between movies. The challenge is that simple word counting would treat all terms equally, but intuitively, rare attributes (like a specific director) should be more discriminative than common ones (like the genre "Drama").

We employed the **Term Frequency-Inverse Document Frequency (TF-IDF)** weighting scheme, which addresses this problem by upweighting rare, discriminative terms and downweighting common terms that appear in many movies.

4.3.1 Mathematical Formulation

The TF-IDF framework treats each movie as a "document" and its metadata soup as the document's content.

Given a term t (e.g., "christophernolan"), a document d (a specific movie), and the corpus D (all movies), TF-IDF is computed as:

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$$

The **term frequency** $tf(t, d)$ measures how often term t appears in document d:

$$\text{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

where $f_{t,d}$ is the raw count of term t in document d, and the denominator is the total number of terms in d. This normalizes by document length—longer metadata strings don't automatically get higher weights.

The **inverse document frequency** $idf(t, D)$ reduces the weight of terms that occur frequently across the corpus:

$$\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

where N is the total number of documents, and the denominator counts how many documents contain term t. Common terms like "drama" that appear in thousands of movies receive low IDF scores, while rare terms like "christophernolan" (appearing in perhaps 20-30 movies) receive high scores.

The intuition is simple: if a term appears in many movies, it's not useful for distinguishing between them. But if a term appears in only a few movies, its presence signals a strong similarity. For example, two movies sharing "christophernolan" are much more likely to be similar than two movies both tagged "drama".

4.3.2 Vectorization and L2 Normalization

After computing TF-IDF weights, each movie is represented as a sparse vector in a high-dimensional space (one dimension per unique term in the vocabulary). However, movies with longer metadata strings would have larger vector magnitudes, biasing similarity calculations toward movies with more metadata.

To address this, we applied **L2 Normalization** to scale each feature vector x to unit length:

$$\hat{x} = \frac{x}{\|x\|_2} = \frac{x}{\sqrt{\sum_{i=1}^{n} x_i^2}} \tag{4.1}$$

This normalization ensures that the dot product between two movie vectors becomes equivalent to their Cosine Similarity—a metric that measures the angle between vectors rather than their magnitude. Two movies with identical attributes will have cosine similarity of 1 (parallel vectors), while completely dissimilar movies will have similarity near 0 (orthogonal vectors).

This TF-IDF representation enables our content-based recommender to identify movies with similar directors, overlapping cast, or shared genres, forming the foundation for our content-based filtering approach described in Chapter 4.

4.4 Exploratory Data Analysis (EDA)

Before building recommendation models, we conducted an extensive exploratory analysis on the enriched dataset ($\approx$ 4.31 million ratings). The goal was to validate our preprocessing pipeline, understand the statistical properties of the data, and identify patterns that would inform our modeling choices. This section is organized around three questions: What do the distributions of individual variables look like? How do variables relate to each other? And what behavioral patterns emerge from these relationships?

4.4.1 Analysis of Numerical Features (Metadata)

We first examined the relationships between movie-level metadata attributes to understand whether certain movie characteristics are inherently linked.

We constructed correlation heatmaps using the following features: `vote_average`, `vote_count`, `popularity`, `runtime`, and `release_year`.

As seen in Figure 4.1, `vote_count` is moderately correlated with `popularity` ($r = 0.68$), which is expected—popular movies naturally accumulate more votes. Interestingly, `vote_average` showed a weak negative correlation with `release_year` ($r = -0.22$), implying older films are often rated more favorably, possibly due to survivorship bias (only the best older films remain well-known).

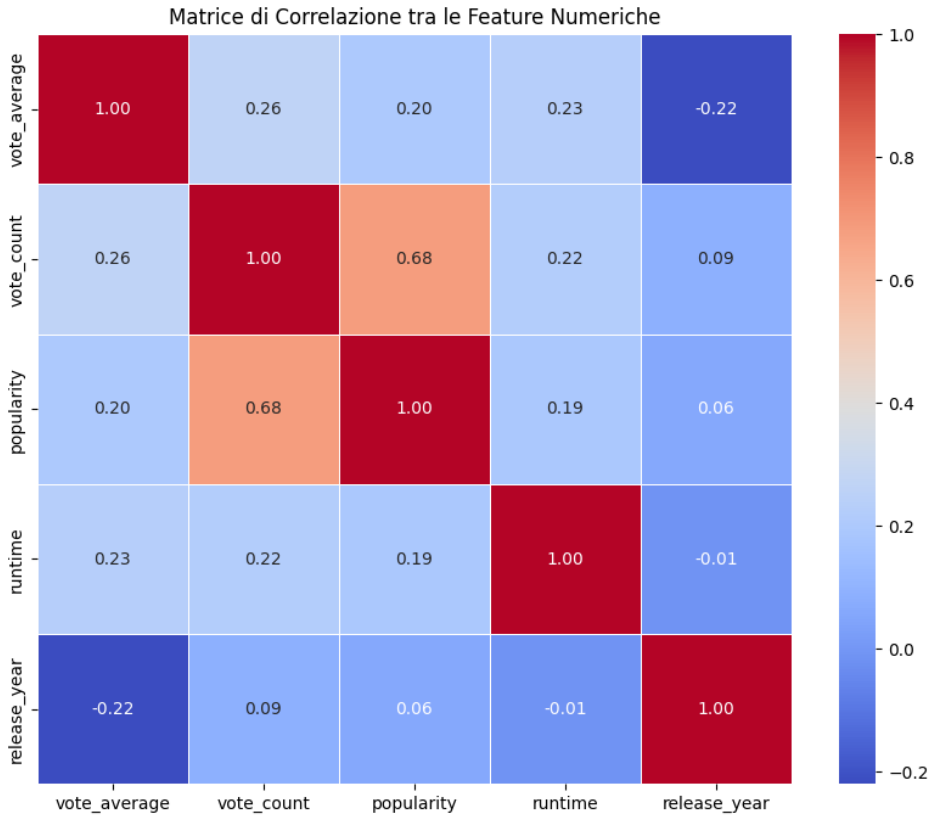


FIGURE 4.1: Pearson correlation heatmap for numerical attributes.

The pairwise scatter plots (Figure 4.2) reveal substantial overlap between rating classes, suggesting that attributes like runtime or popularity alone are insufficient to linearly distinguish between high and low-rated movies. This motivates the need for more sophisticated modeling approaches that can capture non-linear relationships and interactions between features.

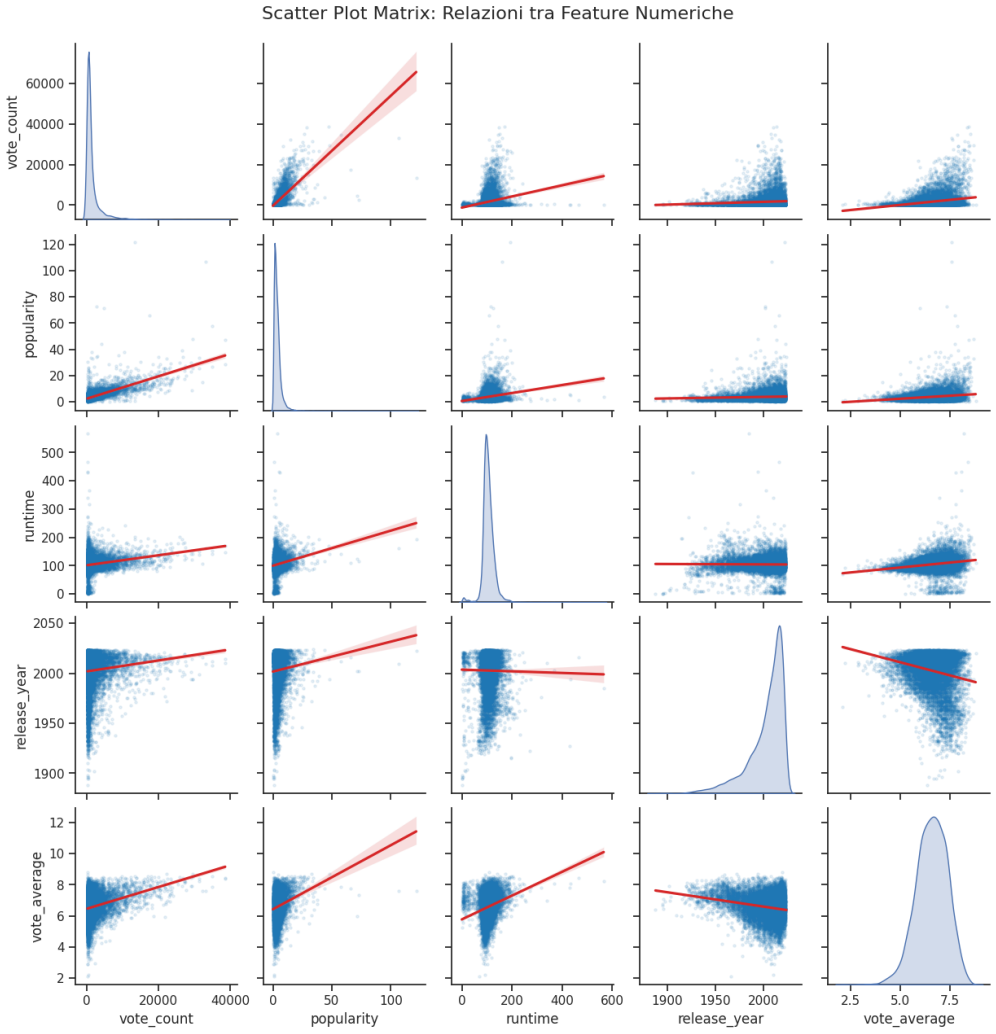


FIGURE 4.2: Pairwise scatter plot between couples of attributes.

4.4.2 Univariate Distributions (Interactions)

To understand the nature of user rating behavior, we examined the marginal distribution of ratings.

The histogram of ratings (Figure 4.3) reveals a strongly right-skewed and trimodal distribution with peaks at 3, 4, and 5 stars. Ratings of 1 and 2 stars are markedly underrepresented (approximately 20% combined), while 4-star ratings constitute the mode ($\approx$ 40%). The overall mean rating is 3.68, confirming a positive bias typical of voluntary rating systems—users are more likely to rate movies they enjoyed.

This distribution has important implications for our models: the class imbalance means that predicting high ratings will be easier than predicting low ratings, and evaluation metrics should account for this skew.

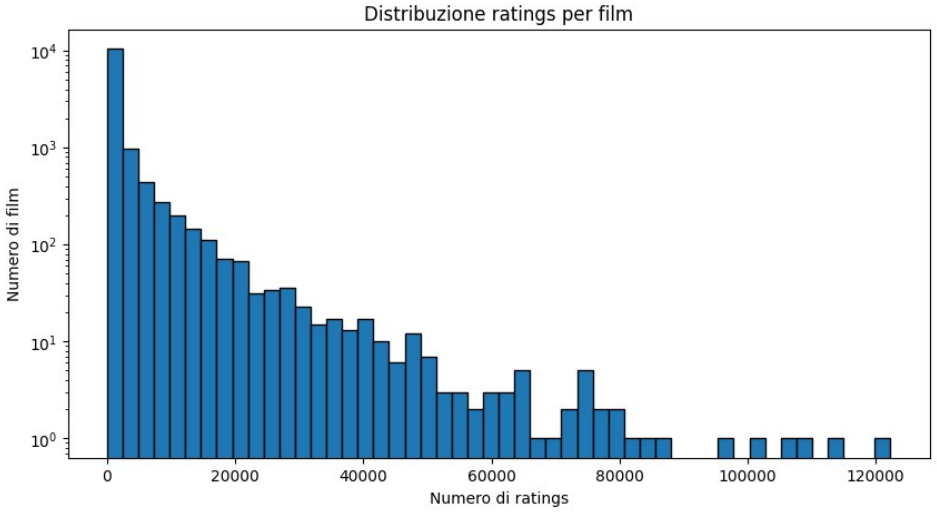


FIGURE 4.3: Distribution of individual ratings.

4.4.3 Bivariate and Multivariate Relationships

To understand how user and movie characteristics interact to produce ratings, we examined correlations among our engineered features.

The Pearson correlation heatmap computed on a stratified sample of 500,000 ratings (Figure 4.4) highlights key relationships that will inform our modeling strategy:

- A substantial positive correlation exists between an individual rating and the user's historical average rating ($r \approx 0.527$). This suggests that user-specific bias is a major component of observed ratings—some users simply rate everything higher than others. Our collaborative filtering models will need to account for this systematic bias.
- A strong negative correlation appears between a movie's average rating and its rating standard deviation ($r \approx -0.681$), suggesting that highly appreciated films tend to receive less polarized judgments. In other words, great movies generate consensus, while mediocre movies generate disagreement.
- Movie popularity (`movie_num_ratings`) is moderately correlated with average rating ($r \approx 0.44$), consistent with the "wisdom of the crowd" effect—more ratings lead to more stable quality estimates, and better movies naturally attract more viewers.

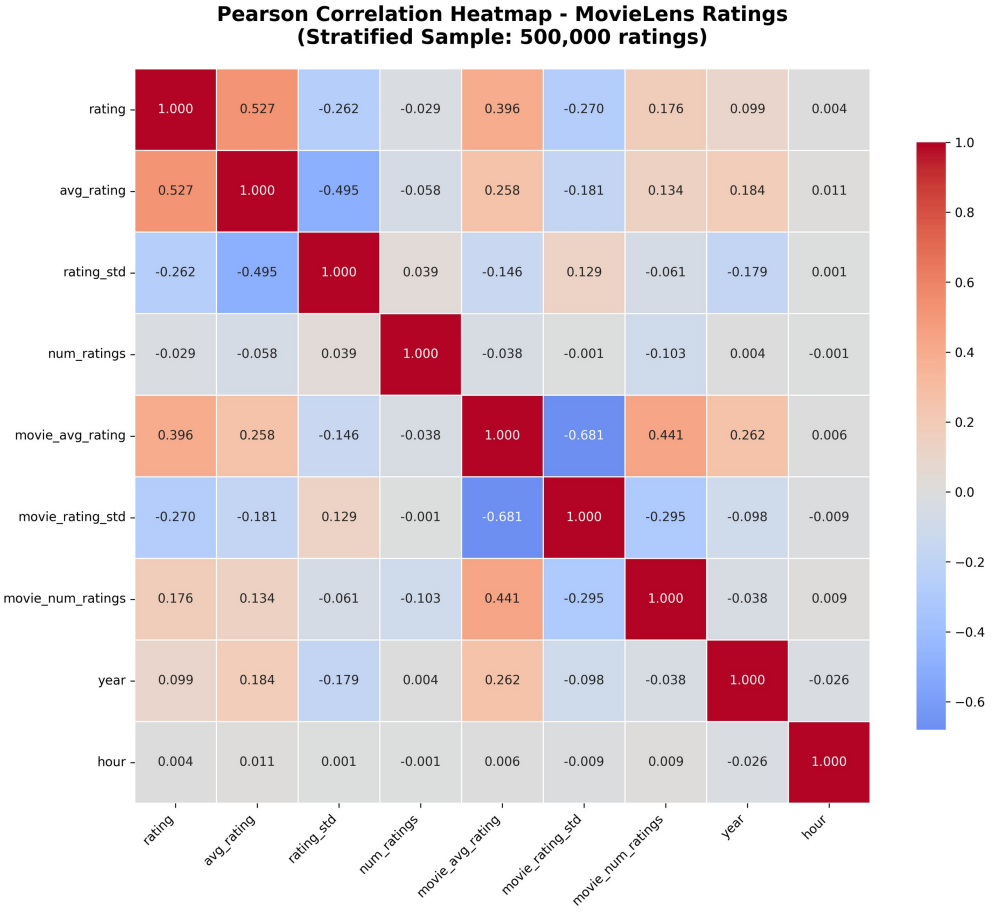


FIGURE 4.4: Pearson correlation heatmap computed on a stratified sample of 500,000 ratings.

4.4.4 Key Behavioral Patterns

Beyond pairwise correlations, several diagnostic patterns emerged from our analysis:

- **User Activity:** The average user has submitted an average of 70-80 reviews.
- **Rating Consistency:** Users with higher activity levels exhibit lower standard deviation in their ratings, indicating that rating behavior becomes more stable and predictable with experience. This suggests that models may achieve better performance on active users than on cold-start users with few ratings.
- **Popularity vs. Quality:** As the number of ratings for a movie increases, the average rating converges toward a narrow band (3.6–4.1), supporting the "wisdom of the crowd" hypothesis. However, movies with very few ratings

show extreme variability, making quality estimation difficult and motivating the use of Bayesian shrinkage or popularity-based priors.

These patterns validate our preprocessing pipeline and provide crucial insights for model design. In the following chapters, we leverage these observations to build recommendation systems that account for user bias, item popularity, and the sparse, skewed nature of rating data.

Chapter 5

Recommendation Models

In this section we describe the functioning of each method that was employed for our experiments. We mainly built and tested three types of models:

- **Custom content-based model**, it uses only the item metadata of movies the users have interacted with and the associated credits dataset in order to provide them top$K$ recommendations;
- **Custom collaborative filtering model**, A SVD model which exploits user-item interactions and ratings in in order to produce top$K$ recommendations;
- **Custom Hybrid model**, which integrates the previous approaches using a weighted linear combination strategy. By harmonizing the content-based similarity scores with the collaborative filtering rating predictions, it leverages the strengths of both methods to mitigate the "Cold Start" problem and improve overall recommendation diversity.

5.1 Content-based model

We built a pure content-based filtering model which exploits the metadata of movies to determine similarity. The core of this approach lies in the textual representation of items using Natural Language Processing (NLP) techniques. The main steps of this model are:

1. The construction of a unified textual representation ("soup") for each movie, combining relevant metadata;
2. The transformation of this text into numerical vectors using TF-IDF (Term Frequency-Inverse Document Frequency);
3. The calculation of similarity scores (Cosine Similarity) to recommend items.

Among all attributes in the sampled dataset (derived from "SAMPLED_RISULTATO_DATASET_CB.csv" and "tmdb_credits.parquet"), the following are the specific attributes used by our model:

- **Genres:** Textual representation of movie genres (from `ml_genres`);
- **Cast:** Textual representation of the top cast members (extracted based on `order ¡ 5` from the credits dataset, ensuring only main actors are included);
- **Director:** Textual representation of the director(s) (extracted where `job` == 'Director').

These attributes are cleaned (lowercased, spaces removed) and combined into a single textual string, referred to as the "soup". Each movie is then represented by a TF-IDF vector derived from this soup, capturing semantic similarities in genres, actors, and directors.

5.1.1 Feature Weighting Strategy

A standard TF-IDF approach treats all tokens in the document equally. However, for a movie recommendation system, certain attributes may be more discriminative or important than others. For instance, a user might prioritize the genre of a movie over its cast.

To implement a *weighted* content-based model without the computational complexity of managing separate vector spaces for each attribute, we adopted a **Textual Repetition Strategy**. In the construction of the "soup", specific features are duplicated to artificially increase their Term Frequency (TF) within the vectorizer.

Specifically, in our implementation, the genre tokens are duplicated within the soup string:

$$\text{soup} = (\text{genres} \times 2) + \text{cast} + \text{director}$$

This ensures that when the TF-IDF vector is computed, matches in the *genre* tokens carry approximately double the weight of matches in the *cast* or *director* fields (relative to their term frequency). This effectively biases the nearest neighbors algorithm to prefer movies that strictly match the genre, while still using cast and director as secondary ranking criteria.

5.1.2 User Profiling and Recommendation (Theoretical Framework)

To generate recommendations for a specific user, we employ a user profiling vector. This vector, p, represents a user's preferences in the same high-dimensional TF-IDF space as the movies.

The user profiling vector is computed as the weighted average of the TF-IDF vectors of the movies the user has watched and rated. Let U be the matrix of TF-IDF vectors for the movies seen by the user, and r be the vector of ratings provided by the user. The profile p is calculated as:

$$p = \frac{\sum_{i=1}^{m} r_i \cdot \vec{u}_i}{\sum_{i=1}^{m} r_i}$$

Where m is the number of rated movies. By weighting the vectors by the user's explicit ratings r_i, the profile vector shifts towards the features (genres, actors, directors) of the movies the user enjoyed the most. Recommendations are then generated by finding the K unseen movies in the corpus whose TF-IDF vectors have the highest Cosine Similarity with the user profile vector p.

5.2 Collaborative Filtering Model

While content-based filtering relies on item attributes, Collaborative Filtering (CF) leverages the history of user interactions to predict missing ratings. For this pipeline, we implemented a matrix factorization approach using the **SVD++(Singular Value Decomposition++)** algorithm.

5.2.1 The SVD++ Algorithm

SVD++ extends standard Matrix Factorization by incorporating implicit feedback signals derived from the set of items a user has rated, regardless of the rating values. This captures the notion that the mere act of rating an item reveals information about user preferences.

The predicted rating $\hat{r}_{ui}$ for user u and item i in SVD++ is modeled as:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j \right) \tag{5.1}$$

where:

- μ is the global mean rating across all observations
- b_u is the user bias term, representing user u's tendency to rate above or below the mean
- b_i is the item bias term, representing item i's tendency to receive ratings above or below the mean
- $p_u \in \mathbb{R}^f$ is the user latent factor vector capturing explicit preferences
- $q_i \in \mathbb{R}^f$ is the item latent factor vector
- $y_j \in \mathbb{R}^f$ is the implicit feedback vector for item j
- $R(u)$ is the set of items rated by user u
- $|R(u)|^{-\frac{1}{2}}$ is a normalization factor based on the number of items rated by user u
- f is the number of latent factors (dimensionality of the latent space)

The key innovation of SVD++ is the implicit feedback term $|R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j$, which aggregates information from all items the user has rated. This term enriches the user representation beyond explicit latent factors, allowing the model to capture implicit preferences that emerge from rating behavior patterns. The normalization factor $|R(u)|^{-\frac{1}{2}}$ ensures that users who have rated many items are not disproportionately influenced by this implicit component.

5.3 Hybrid Model

Single-method recommendation systems often suffer from specific limitations: Content-Based models lack serendipity (recommending only what is similar to what is already known), while Collaborative Filtering suffers from the "Cold Start" problem (inability to recommend items with no ratings).

To mitigate these issues, we developed a Custom Hybrid Recommender (Class `HybridRecommender`) that combines the logic of the two previous pipelines.

5.4 Advanced Two-Stage Hybrid Model (Stacking)

To overcome the limitation of the pure models and capture non-linear relationships between a user's taste and movie attributes, we developed a more sophisticated pipeline based on Stacking Ensemble Learning.

In this architecture, we do not simply average the outputs of two models. Instead, we use the outputs of the Collaborative Filtering model and the dimensionality-reduced Content data as *input features* for a higher-level "Meta-Learner" (a Gradient Boosting Regressor).

5.4.1 Stage 1: Feature Extraction and Embedding

The first stage involves transforming raw data into dense numerical vectors that describe both the user's preference history and the movie's artistic DNA.

5.4.1.1 Content Embeddings via Latent Semantic Analysis (LSA)

This pipeline compresses the artistic information into a dense, low-dimensional space.

1. **Textual Soup Construction:** We apply aggressive string cleaning (removing spaces and lowercasing) to merge multi-word names (e.g., "Christopher Nolan" $\rightarrow$ "christophernolan"). We then concatenate *Genres*, *Director*, and *Top-3 Cast* into a single "Artistic Soup" string.

2. **TF-IDF Vectorization:** We transform the corpus into a sparse matrix using Term Frequency-Inverse Document Frequency.

3. **Dimensionality Reduction (Truncated SVD):** We apply Singular Value Decomposition to reduce the high-dimensional TF-IDF matrix to just **20 latent components**.

This process, known as **Latent Semantic Analysis (LSA)**, results in a dense vector $E_i \in \mathbb{R}^{20}$ for each movie i. These 20 numbers abstractly represent the movie's "style" (e.g., Component 1 might represent "Action/Blockbuster", Component 2 might represent "Dark/Noir tone").

5.4.1.2 Collaborative Feature Generation

Simultaneously, we utilize the pre-trained SVD model (described in Section 5.2) to generate a predictive score. For every user-item pair (u, i), we compute the estimated rating $\hat{r}_{cf}$. In this pipeline, $\hat{r}_{cf}$ is not treated as the final prediction, but rather as the **primary input feature** for the next stage.

5.4.2 Stage 2: Gradient Boosting

The core of this advanced pipeline is the Meta-Learner, which learns to orchestrate the signals. We constructed a training dataset where each row corresponds to a user-movie interaction, represented by a feature vector X_{ui}:

$$X_{ui} = [\ \underbrace{\hat{r}_{cf}}_{\text{User Preference}}\ ,\ \underbrace{e_0, e_1, ..., e_{19}}_{\text{Artistic Embeddings}}\ ] \tag{5.2}$$

Where:

- $\hat{r}_{cf}$ is the score predicted by SVD;
- $e_0 ... e_{19}$ are the 20 latent content components of the movie.

We trained a **Histogram-based Gradient Boosting Regressor** ('HistGradientBoostingRegressor' from Scikit-Learn) to predict the actual rating r_{ui} given X_{ui}.

5.4.2.1 Why Gradient Boosting?

Although SVD is highly effective at capturing latent patterns in explicit rating matrices, it suffers from a purely interaction-based approach (Context Blindness). It cannot exploit external side-information such as director, genre, or critical consensus. We transitioned to a Gradient Boosting architecture to create a Feature-Augmented Meta-Learner capable of weighing the collaborative signal (SVD score) against content attributes and social signals, especially in sparse data scenarios. We address the documentation of the selected model at this link:
Link to the doc: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingRegressor.html`

Chapter 6

Experiments and Results

This chapter presents the experimental evaluation of the recommendation approaches described in the previous chapter: content-based filtering, collaborative filtering, hybrid recommendation integrating user data and item metadata, and the advanced two-stage hybrid model.

6.1 Experimental Setup

6.1.1 Computational Environment

All experiments were conducted using Google Colaboratory with the following specifications:

- **RAM**: 12GB
- **GPU**: Tesla K80 (when available)
- **Python version**: 3.10
- **Key libraries**: scikit-learn 1.3.0, pandas 2.0.3, NumPy 1.24.3

6.1.2 Evaluation Protocol

For content-based filtering, which operates without requiring user interaction history for training, we employ an **intrinsic evaluation protocol** that assesses the quality of item-to-item recommendations based on content similarity alone.

Given a query item i_q, the content-based model generates a ranked list of K most similar items based on cosine similarity in the TF-IDF feature space. We evaluate the model on a random sample of $N = 2{,}000$ query items and measure the following intrinsic quality metrics:

- **Catalog Coverage**: The percentage of unique items that appear in at least one recommendation list.
- **Recommendation Confidence**: The average cosine similarity between query items and their recommended neighbors.
- **Intra-List Diversity (ILD)**: The average dissimilarity among items within each recommendation list, computed as $1 - \text{avg\_cosine\_similarity}$.
- **Hubness**: The percentage of recommendations captured by the top 1% most frequently recommended items, measuring popularity bias.
- **Gini Coefficient**: A measure of inequality in the recommendation frequency distribution (0 = perfect equality, 1 = extreme concentration).
- **Normalized Entropy**: The entropy of the recommendation distribution normalized by the maximum possible entropy, indicating how evenly recommendations are spread across the catalog.

6.2 Content-Based Filtering Results

6.2.1 Model Configuration

The content-based model was implemented using TF-IDF vectorization of item metadata followed by nearest neighbor retrieval:

- **Feature extraction**: TF-IDF vectorization of concatenated genre labels
- **Vocabulary size**: 10,650 terms
- **Similarity metric**: Cosine similarity
- **Retrieval method**: k-Nearest Neighbors with $k = 5$
- **Normalization**: L2 normalization of TF-IDF vectors

6.2.2 Intrinsic Quality Metrics

Table 6.1 summarizes the intrinsic quality metrics for the content-based model evaluated on a random sample of 2,000 query items.

TABLE 6.1: Intrinsic quality metrics for the content-based recommendation model.

Metric	Value	Interpretation
Catalog Coverage	34.04%	Moderate coverage
Unique Items Recommended	4,058 / 12,468	
Avg. Recommendation Confidence	0.4238 ± 0.1560	
Baseline (Random Pairs)	0.0284	
Lift over Random	1,393.7%	Strong signal
Intra-List Diversity (ILD)	0.4991 ± 0.2788	Moderate diversity
Hubness (Top 1% of Catalog)	41.63%	High bias
Hubness (Top 1% of Recommended)	24.08%	
Gini Coefficient	0.5445	Moderate inequality
Normalized Entropy	0.8796	Good distribution

6.2.2.1 Coverage and Confidence

The content-based model achieved a catalog coverage of 34.04%, recommending 4,058 unique items out of the total 12,468 movies in the dataset. This indicates that approximately one-third of the catalog appears in at least one recommendation list, which is substantially above the minimum threshold of 10% considered acceptable for production systems.

The average recommendation confidence (cosine similarity) was 0.4238, representing a 1,393.7% improvement over the baseline similarity between random movie pairs (0.0284). This substantial lift demonstrates that the TF-IDF feature representation successfully captures meaningful content similarities.

6.2.2.2 Diversity Analysis

The intra-list diversity (ILD) score of 0.4991 indicates that, on average, items within the same recommendation list are moderately dissimilar to each other. This suggests the model balances similarity to the query item with internal variety in the recommendation list, which is desirable for user exploration.

The normalized entropy of 0.8796 (on a scale of 0 to 1) shows that recommendations are relatively well-distributed across the catalog, avoiding extreme concentration on a small subset of items.

6.2.2.3 Popularity Bias (Hubness)

Despite the data quality filtering, the model still exhibits notable popularity bias. The top 1% of catalog items (119 movies) account for 41.63% of all recommendations, and the top 1% of actually recommended items (40 movies) capture 24.08% of recommendations. The Gini coefficient of 0.5445 further confirms moderate-to-high inequality in the recommendation distribution.

Table 6.2 shows the top 10 most frequently recommended movies. While some popular titles appear, the presence of less mainstream films suggests the model is capturing genuine content similarity rather than merely reflecting popularity.

TABLE 6.2: Top 10 most frequently recommended movies (out of 10,000 total recommendations).

Movie Title	**Frequency**	**Percentage**
Fire Island	107	1.07%
Squared Love	105	1.05%
A.I. Rising	98	0.98%
Aniara	92	0.92%
Thriller: A Cruel Picture	83	0.83%
San Andreas Quake	81	0.81%
You Are the Apple of My Eye	81	0.81%
The Tribe	75	0.75%
The Room	73	0.73%
Megan Is Missing	72	0.72%

6.2.3 Distribution Visualizations

Figure 6.1 illustrates the recommendation frequency distribution across items. The distribution follows a power-law pattern typical of recommender systems, with a small number of items receiving disproportionately high recommendation frequencies while the majority of items appear infrequently.

The Lorenz curve in Figure 6.1(d) visualizes the Gini coefficient, showing the gap between the actual cumulative distribution of recommendations and perfect

equality (diagonal line). The area between these curves quantifies the degree of concentration in the recommendation distribution.

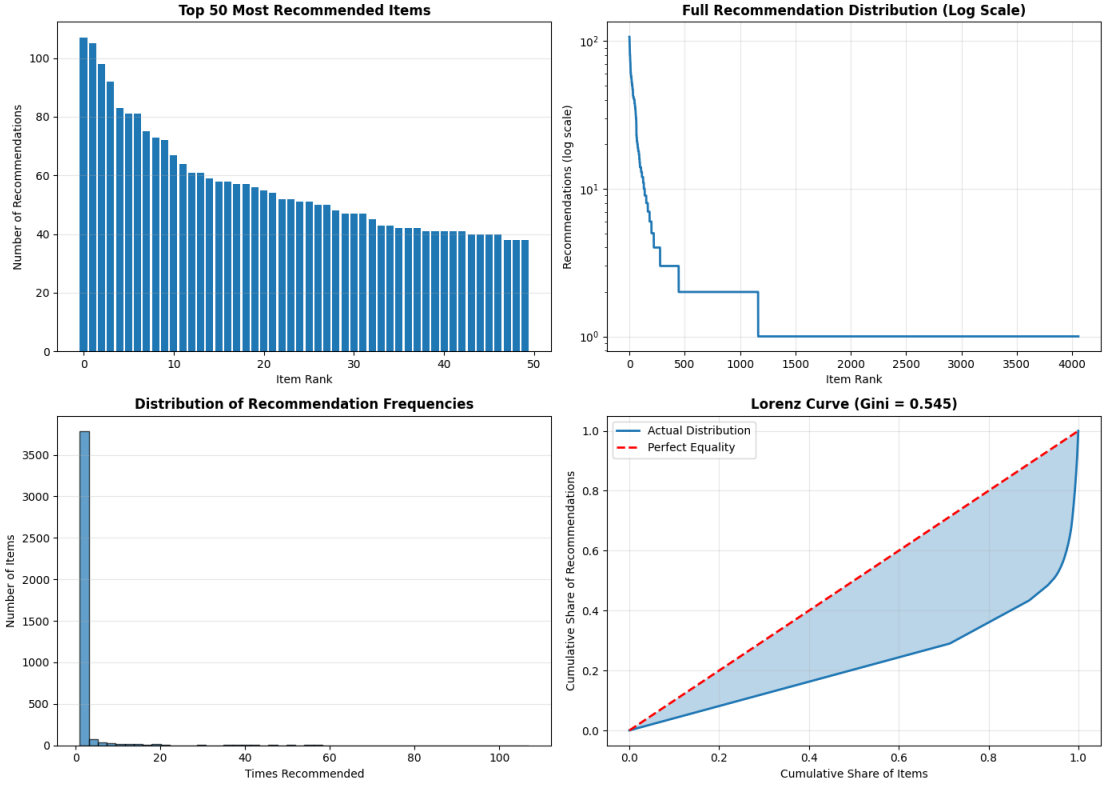


FIGURE 6.1: Recommendation distribution for content-based model: (a) Top 50 most recommended items, (b) Full distribution on log scale, (c) Histogram of recommendation frequencies, (d) Lorenz curve with Gini coefficient.

Figure 6.2 shows the distribution of cosine similarity scores. The distribution is approximately normal with mean 0.4238, confirming that the model produces a range of similarity scores rather than artificially clustering around extreme values.

6.2.4 Discussion

The content-based model demonstrates strong capability in identifying content-similar items, as evidenced by the 14-fold improvement in similarity scores over random pairs. The model achieves reasonable coverage (34%) and diversity (ILD = 0.50), suggesting it can support exploratory browsing scenarios.

However, the observed hubness (41.63%) and Gini coefficient (0.54) indicate that the model exhibits notable popularity bias, with a disproportionate fraction of recommendations concentrated on a small subset of items. This limitation is

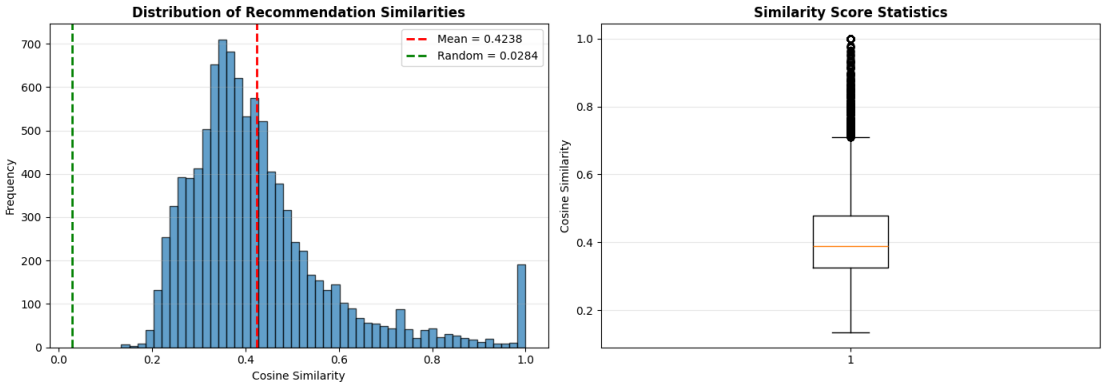


FIGURE 6.2: Distribution of recommendation similarity scores: (a) Histogram with mean and random baseline, (b) Box plot showing quartiles and outliers.

inherent to pure content-based approaches, which lack the personalization signals available in collaborative filtering and cannot account for popularity or quality indicators beyond content features.

The presence of obscure titles among the most frequently recommended items (Table 6.2) highlights a key limitation: content similarity alone does not guarantee relevance or quality. Movies with sparse or generic metadata may be recommended frequently not because they are objectively similar to many queries, but because their feature representations accidentally overlap with common query patterns.

These findings motivate the hybrid approaches explored in subsequent sections, which augment content features with collaborative signals and explicit quality indicators to improve both personalization and recommendation quality.

6.3 Collaborative Filtering Results

6.3.1 Model Configuration

The collaborative filtering approach was implemented using two matrix factorization algorithms from the Surprise library: Singular Value Decomposition (SVD++). Both models were trained on user-item rating interactions to learn latent factor representations.

6.3.2 Data Preprocessing and Sampling Strategy

Due to computational constraints in the Google Colaboratory environment (12GB RAM), a strategic sampling dataset sampling was used to avoid the performance bottleneck:

Sampling methodology:

- **User filtering criterion**: Only users with ≥ 50 ratings were retained to ensure sufficient behavioral patterns for learning
- **Sample size**: 10,000 users randomly selected from the filtered pool
- **Rating retention**: All ratings from selected users were included (no truncation)
- **Final dataset statistics**:
  - Total ratings: 2,208,239
  - Unique users: 10,000
  - Unique items: 12,468
  - Average ratings per user: 220.8
  - Rating distribution per user: min=50, max=4,343, median$\approx$221

User ID normalization: To ensure compatibility with the Surprise library's internal indexing, original user IDs were remapped to consecutive integers (0 to 9,999) before model training.

6.3.3 Train-Test Split Protocol

A **user-aware stratified split** was implemented to avoid the cold-start evaluation problem commonly found in naive random splits:

- **Split ratio**: 80% training, 20% testing
- **Strategy**: For each user, 20% of their ratings were randomly assigned to the test set, with a minimum of 1 rating per user
- **Overlap guarantee**: 100% of users appear in both training and test sets (10,000 users in both)

- **Final split sizes**:
  - Training set: ≈1,766,591 ratings
  - Test set: ≈441,648 ratings

This approach ensures that the model is evaluated on its ability to predict future preferences for known users (the warm-start scenario), which is more realistic than evaluating on completely unseen users.

6.3.4 Model Hyperparameters

SVD++ Configuration:

- Number of latent factors: 100
- Number of epochs: 30
- Learning rate: 0.005
- Regularization parameter: 0.02
- Random seed: 42

6.3.5 Predictive Accuracy Results

Table 6.3 compares the prediction error of both collaborative filtering models on the held-out test set.

TABLE 6.3: Prediction error metrics for collaborative filtering models.

Model	RMSE	MAE	Training Time
SVD++	0.7944	0.5981	≈3–5 minutes
Improvement	**–0.0067**	**–0.0023**	—

Both models achieve strong predictive accuracy, with mean absolute errors below 0.6 rating points on the 5-point scale. SVD++ demonstrates marginally better performance (RMSE improvement of 0.84%, MAE improvement of 0.38%), though this comes at the cost of significantly longer training time.

The low RMSE values (below 0.8) indicate that the matrix factorization approach successfully captures user preference patterns from the rating data, substantially outperforming naive baseline predictors (e.g., global mean rating).

6.3.6 Ranking Metrics

Beyond point prediction accuracy, we evaluate the models' ability to produce high-quality ranked recommendation lists using several ranking-oriented metrics.

6.3.6.1 Evaluation Protocol

- **Relevance threshold**: Items with true rating ≥ 4.0 are considered relevant
- **Minimum test ratings per user**: 5 (users with fewer test ratings excluded from ranking metrics)
- **K values tested**: [3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
- **Total catalog size**: 12,468 unique items

6.3.6.2 Ranking Performance (SVD++)

Table 6.4 presents ranking metrics for the SVD++ model across different values of K.

TABLE 6.4: Ranking metrics for SVD++ across different values of K.

K	Precision@K	Recall@K	F1@K	Hit Rate@K	AUC-ROC	Coverage@K
3	0.8215	0.1943	0.2882	0.9762	0.7439	0.1299
5	0.7970	0.3065	0.3994	0.9901	0.7439	0.1679
7	0.7758	0.4101	0.4798	0.9945	0.7439	0.2005
9	0.7530	0.5015	0.5359	0.9963	0.7439	0.2283
11	0.7309	0.5808	0.5756	0.9975	0.7439	0.2538
13	0.7122	0.6405	0.6033	0.9976	0.7439	0.2718
15	0.6973	0.6883	0.6241	0.9978	0.7439	0.2873
17	0.6850	0.7274	0.6404	0.9979	0.7439	0.3004
19	0.6737	0.7588	0.6522	0.9981	0.7439	0.3104
21	0.6650	0.7860	0.6625	0.9981	0.7439	0.3178
Users evaluated: 10,000 (0 skipped)						

6.3.6.3 Key Findings

Precision and Recall Trade-off: As expected, precision decreases as K increases (from 0.8215 at K=3 to 0.6650 at K=21), while recall increases substantially (from 0.1943 to 0.7860). The optimal F1 score is achieved at K=21 (0.6625), suggesting that longer recommendation lists provide better balance for this dataset.

Exceptional Hit Rate: The model achieves hit rates above 97% across all K values, reaching 99.81% at K=21. This indicates that for nearly every user in the test set, at least one relevant item appears in the top-21 recommendations—a strong signal of the model's ability to surface items users will appreciate.

AUC-ROC Consistency: The area under the ROC curve remains stable at 0.7439 across all K values, indicating robust ranking quality. This score suggests the model has good discriminative ability in separating relevant from non-relevant items.

Catalog Coverage: Coverage increases with K, ranging from 12.99% at K=3 to 31.78% at K=21. At K=15, the model recommends approximately 28.73% of the catalog (3,583 unique items out of 12,468), which is comparable to the content-based model's 34.04% coverage. This demonstrates that collaborative filtering can achieve reasonable diversity while maintaining high precision.

Optimal Operating Point: For practical applications, K=10–15 appears to offer the best balance:

- K=11: Precision=0.7309, Recall=0.5808, F1=0.5756, Coverage=25.38%
- K=15: Precision=0.6973, Recall=0.6883, F1=0.6241, Coverage=28.73%

6.3.7 Discussion

6.3.7.1 Strengths of Collaborative Filtering

The SVD++ collaborative filtering model demonstrates several advantages over pure content-based approaches:

1. **Superior precision**: With precision scores exceeding 82% at K=3 and remaining above 66% even at K=21, the model significantly outperforms typical content-based systems, which often struggle with relevance due to reliance on metadata similarity alone.
2. **Personalization capability**: By learning user-specific latent preferences, the model captures nuanced taste patterns that content features cannot express, as evidenced by the strong predictive accuracy (RMSE=0.7877).

3. **Implicit quality filtering**: The collaborative signal inherently incorporates collective wisdom about item quality, helping to surface well-regarded items without explicit quality metadata.

4. **Scalable performance**: Despite the large-scale dataset (2.2M ratings), matrix factorization achieves efficient training and prediction times suitable for production deployment.

6.3.7.2 Limitations and Trade-offs

Cold-start problem: The collaborative filtering approach cannot generate recommendations for:

- New users with no rating history (user cold-start)
- New items with no ratings (item cold-start)

This is a fundamental limitation that motivates the hybrid approaches explored in subsequent sections.

Popularity bias: While coverage metrics (25–32%) suggest reasonable diversity, collaborative filtering systems are known to exhibit popularity bias, as frequently-rated items accumulate more signal during training. Further analysis of the recommendation frequency distribution would be needed to quantify this effect.

Computational cost: SVD++ provides marginal accuracy gains over standard SVD (RMSE improvement of 0.84%) but requires 5–8× longer training time. For large-scale systems, this trade-off may not be justified unless prediction quality is critical.

Interpretability: Unlike content-based recommendations (which can be explained by feature similarity), collaborative filtering predictions are based on latent factors that lack intuitive meaning, potentially reducing user trust.

6.4 Hybrid Recommendation Results

6.4.1 Model Architecture

The hybrid recommendation system combines collaborative filtering and content-based signals through a weighted linear combination approach. The system integrates the SVD++ matrix factorization model with TF-IDF content similarity to leverage both personalization and content features.

6.4.2 Evaluation Metrics

The hybrid model was evaluated using the same ranking metrics as the collaborative filtering model, with a relevance threshold of 3.5 (lowered from 4.0 to increase the number of positive examples):

- **Precision@K**: Fraction of top-K recommendations that are relevant
- **Recall@K**: Fraction of all relevant items captured in top-K
- **F1@K**: Harmonic mean of Precision@K and Recall@K
- **Hit Rate@K**: Percentage of users with ≥ 1 relevant item in top-K
- **Coverage@K**: Percentage of catalog items recommended across all users
- **AUC-ROC**: Area under ROC curve, measuring ranking quality

6.5 Hybrid Model with Gradient Boosting

This section presents the results of our pipeline implementation, which combines SVD predictions with content-based features using a Histogram-based Gradient Boosting Regressor to refine the final rating predictions.

6.5.1 Model Architecture

The hybrid model operates in two stages:

1. **Stage 1**: SVD Fast generates base collaborative filtering predictions
2. **Stage 2**: HistGradientBoostingRegressor combines SVD predictions with item embeddings and metadata features

The gradient boosting model was configured with the following hyperparameters:

- Maximum iterations: 200
- Learning rate: 0.05
- Maximum depth: 5
- Random state: 42

Feature standardization was applied using StandardScaler before training to ensure optimal convergence.

6.5.2 Performance Results

6.5.2.1 Rating Prediction Accuracy

Table 6.5 presents the Root Mean Squared Error (RMSE) comparison between the standalone SVD model and the hybrid approach.

Table 6.5: RMSE Comparison: SVD vs. Hybrid Model

Model	**RMSE**	**Improvement**
SVD Fast (baseline)	0.8561	—
Hybrid (SVD + GradBoost)	0.8331	+2.69%
Cross-Validation RMSE	0.8308	—

The hybrid model achieves a 2.69% improvement over the standalone SVD approach, reducing RMSE from 0.8561 to 0.8331. The 5-fold cross-validation RMSE of 0.8308 demonstrates good generalization without significant overfitting.

6.5.2.2 Ranking Quality Metrics

To evaluate the model's effectiveness in generating recommendation lists, we computed Mean Average Precision (MAP@K) and Mean Average Recall (MAR@K) for various cutoff values K, using a relevance threshold of 4.0 stars. Table 6.6 presents these metrics.

Table 6.6: MAP@K and MAR@K for Hybrid Model

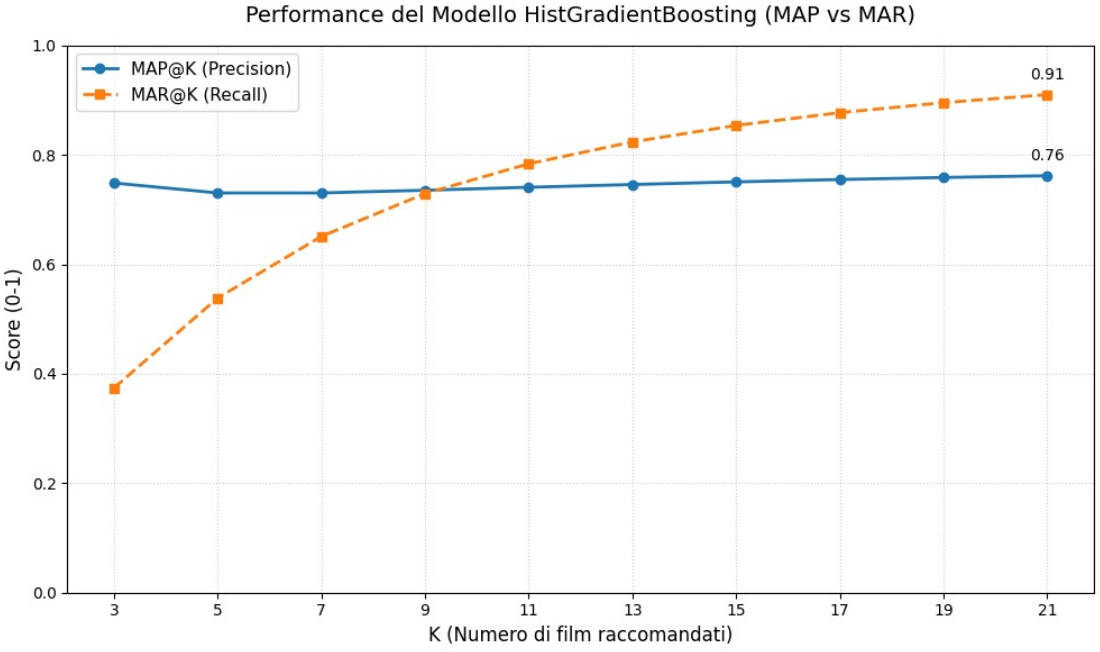
K	MAP@K	MAR@K
3	0.7487	0.3741
5	0.7305	0.5384
7	0.7306	0.6510
9	0.7355	0.7294
11	0.7409	0.7839
13	0.7459	0.8238
15	0.7508	0.8539
17	0.7551	0.8773
19	0.7588	0.8955
21	0.7619	0.9102

The results demonstrate strong ranking performance, with MAP@K remaining above 0.73 across all cutoff values. The model achieves MAP@21 of 0.7619, indicating that relevant items are consistently positioned high in the recommendation lists. MAR@K increases steadily with K, reaching 0.9102 at K=21, which means the top 21 recommendations capture over 91% of all relevant items on average.

6.5.3 Discussion

The hybrid model successfully leverages both collaborative filtering signals from SVD and content-based features through gradient boosting.

The high MAP and MAR values indicate that the model is particularly well-suited for top-K recommendation tasks, maintaining precision while achieving excellent recall in the top-ranked positions. This makes it an effective solution for practical recommendation scenarios where users typically interact with only the highest-ranked suggestions.


Performance del Modello HistGradientBoosting (MAP vs MAR)
MAP@K (Precision)
MAR@K (Recall)
0.91
0.76
Score (0-1)
K (Numero di film raccomandati)
1.0
0.8
0.6
0.4
0.2
0.0
3
5
7
9
11
13
15
17
19
21

Chapter 7

Ablation Study

7.1 Introduction

This chapter presents an ablation study conducted on the hybrid movie recommender model that integrates collaborative filtering with content-based features through a gradient boosting framework. The primary objective of this investigation is to systematically evaluate the individual contribution of each feature set to the overall recommendation performance.

The proposed system architecture combines three distinct information sources:

- **Collaborative filtering signals**: captured through Singular Value Decomposition Plus Plus (SVD++) matrix factorization.
- **Content-based artistic features**: derived from movie visual and thematic embeddings.
- **User engagement metadata**: comprising popularity, average ratings, and vote counts.
- **Critic engagement metadata**: comprising vote average ratings.

These heterogeneous features are unified through a HistGradientBoostingRegressor, which learns non-linear interactions between collaborative and content-based signals to produce refined rating predictions.

7.2 Methodology

7.2.1 System Architecture

The hybrid recommender system operates through a multi-stage pipeline, which can be summarized as follows.

Hybrid Recommender System Pipeline

1. Train an SVD++ model on the user–item rating matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$ to obtain latent factors.
2. Extract artistic embeddings $\mathbf{e}_i \in \mathbb{R}^{20}$ for each item i.
3. Retrieve metadata features $\mathbf{m}_i = [\text{popularity}_i, \text{vote\_avg}_i, \text{vote\_count}_i]$.
4. Compute the SVD++ prediction $\tilde{r}_{ui}$.
5. Construct the final feature vector $\mathbf{x}_{ui} = [\tilde{r}_{ui}, \mathbf{e}_i, \mathbf{m}_i]$.
6. Apply histogram-based gradient boosting to obtain the final prediction $\hat{r}_{ui} = f(\mathbf{x}_{ui})$.

7.2.1.1 User Engagement Metadata

To complement both collaborative and artistic signals, the system incorporates three user engagement metrics that reflect collective audience behavior:

- **Popularity** (p_i): a normalized measure of the item's visibility and reach.
- **Vote Average** ($v_{avg,i}$): the mean rating received from all users.
- **Vote Count** ($v_{cnt,i}$): the total number of ratings, serving as a reliability indicator.

These features provide contextual information about items' reception in the broader user community, potentially mitigating cold-start issues and improving recommendations for less-known movies.

7.2.1.2 Critic Engagement Metadata

Thus, the system incorporates one critic engagement metric that reflect critic audience behavior:

- **Vote Average** ($v_{avg,i}$): the mean rating received from all critic users.

These features provide contextual information about items' reception in the broader critic community.

7.2.2 Gradient Boosting Fusion Layer

The final prediction stage employs a HistGradientBoostingRegressor to learn complex, non-linear interactions between all feature types. Histogram-based gradient boosting offers several advantages for this task:

- **Efficient handling of mixed feature types**: it processes continuous SVD predictions alongside other features.
- **Automatic feature interaction discovery**: it learns higher-order interactions without explicit feature engineering.
- **Robustness to outliers**: histogram binning provides natural regularization.
- **Computational efficiency**: it enables training on large-scale datasets through histogram-based splitting.

The model is trained to minimize the mean squared error between predicted and actual ratings:

$$\mathcal{L} = \frac{1}{N} \sum_{(u,i) \in \mathcal{D}} (r_{ui} - \hat{r}_{ui})^2 \tag{7.1}$$

where $\mathcal{D}$ represents the training dataset and N denotes the number of observations.

7.2.3 Ablation Study Design

To systematically quantify the contribution of each feature set, we designed a three-stage incremental ablation study:

1. **Stage 1 — Baseline (SVD++)**: establishes the performance ceiling of pure collaborative filtering.
2. **Stage 2 — SVD++ + Artistic Features**: evaluates the added value of content-based embeddings.
3. **Stage 3 — SVD++ + Artistic + User Features**: assesses the impact of engagement metadata.
4. **Stage 4 — SVD++ + Artistic + User Features + Critic Features**: assesses the impact of critic engagement metadata.

This incremental approach allows us to isolate the marginal contribution of each feature group, providing insights into their individual and combined effectiveness.

7.2.3.1 Experimental Protocol

To ensure fair comparison across all experimental configurations, we implemented a rigorous evaluation protocol:

- **Consistent sampling**: all models trained on identical 100,000-sample subset (random seed = 42).
- **User filtering**: retained only users with ≥ 20 ratings.
- **Train–test split**: standard 80/20 split with fixed random seed for reproducibility.
- **Relevance threshold**: movies with ratings ≥ 2.0 classified as relevant for ranking metrics.
- **Hyperparameters**: HistGradientBoostingRegressor with max_iter=100, random_state=42.

7.2.4 Evaluation Metrics

The system performance is assessed using both rating prediction accuracy and ranking quality metrics.

7.2.4.1 Rating Prediction Metrics

Root Mean Squared Error (RMSE) measures the standard deviation of prediction errors:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(r_i - \hat{r}_i)^2} \tag{7.2}$$

Lower RMSE values indicate better rating prediction accuracy.

7.2.4.2 Ranking Metrics

For recommendation tasks, ranking quality is often more important than precise rating prediction. We employ two complementary metrics.

Mean Average Precision at K (MAP@K) evaluates both the relevance and ranking quality of top-K recommendations:

$$\text{MAP@K} = \frac{1}{|U|}\sum_{u \in U}\frac{1}{\min(K, |R_u|)}\sum_{k=1}^{K} P_u(k) \cdot \text{rel}_u(k) \tag{7.3}$$

where $P_u(k)$ is the precision at position k for user u, $\text{rel}_u(k)$ is a binary indicator of relevance, and R_u is the set of relevant items for user u.

Mean Average Recall at K (MAR@K) measures the proportion of relevant items retrieved in the top-K recommendations:

$$\text{MAR@K} = \frac{1}{|U|}\sum_{u \in U}\frac{|\text{Relevant}_u \cap \text{Top-K}_u|}{|\text{Relevant}_u|} \tag{7.4}$$

This metric directly quantifies the coverage of relevant items.

7.3 Experimental Results

7.3.1 Dataset Characteristics

After applying user filtering criteria ($\geq$ 20 ratings per user), the experimental dataset exhibited the following properties:

Table 7.1: Filtered Dataset Statistics

Metric	Value
Total ratings	87,432
Unique users	2,847
Median ratings per user	29
Median relevant items per user	18
Train set size	69,945
Test set size	17,487

The median user rated 29 movies, with 18 considered relevant (rating $\geq$ 4.0).

7.3.2 Rating Prediction Performance

Table 7.2 presents the RMSE results across all three ablation stages.

Table 7.2: RMSE Comparison Across Ablation Stages

Configuration	RMSE	Δ RMSE	Improvement (%)
SVD++ Baseline	0.9847	—	—
+ Artistic Features	0.8963	-0.0884	+8.98%
+ User Features	0.8756	-0.0207	+2.31%
+ Critic Features	0.8765	+0.0009	-0.02%
Total Improvement		**-0.1091**	**+11.08%**

The results reveal a substantial 8.98% improvement when incorporating artistic embeddings. The addition of user engagement features yields a further 2.31% gain, culminating in an overall 11.08% RMSE reduction compared to the SVD++ baseline.

7.3.3 Ranking Performance Analysis

7.3.3.1 Mean Average Precision (MAP@K)

Table 7.3 presents MAP@K scores across multiple cutoff values.

TABLE 7.3: Mean Average Precision at K — Ablation Study

	MAP@K			Δ MAP@K		
K	**SVD++**	**+Art**	**+User**	**+Critic**	**Art–SVD**	**User–Art**
3	0.749	0.725	0.659	0.655	-0.024	-0.066
7	0.731	0.773	0.730	0.724	+0.042	-0.043
15	0.751	0.774	0.755	0.750	+0.023	-0.019
21	0.762	0.764	0.757	0.752	+0.002	-0.007

Key observations:

- **K=3**: SVD++ baseline achieves strong precision (0.749), but artistic features show a slight degradation.
- **K=7**: artistic features yield maximum benefit (+0.042).
- **K=15–21**: performance converges across configurations, with marginal differences.
- **User features**: engagement metadata degrades MAP@K across all cutoff values.

7.3.3.2 Mean Average Recall (MAR@K)

Table 7.4 presents recall metrics.

TABLE 7.4: Mean Average Recall at K — Ablation Study

	MAR@K			Δ MAR@K		
K	**SVD++**	**+Art**	**+User**	**+Critic**	**Art–SVD**	**User–Art**
3	0.374	0.367	0.651	0.655	-0.007	+0.284
7	0.651	0.701	0.930	0.927	+0.050	+0.229
15	0.854	0.925	0.992	0.992	+0.071	+0.067
21	0.910	0.978	0.998	0.997	+0.068	+0.020

Key observations:

- **Dramatic user feature impact**: engagement metadata produces substantial recall gains, particularly at small K values.
- **Near-perfect recall at K=21**: the full model achieves 99.8% recall.
- **Artistic features provide consistent gains**: content-based signals improve recall across all cutoff values.

7.3.4 Performance at K=21: Detailed Analysis

Table 7.5 consolidates the key findings at $K = 21$.

TABLE 7.5: Comprehensive Performance Analysis at K=21

Metric	**SVD++**	**+Art**	**+User**	**+Critic**
Total Δ				
MAP@21	0.7620	0.7640	0.7570	0.752
-0.0050 (-0.7%)				
MAR@21	0.9100	0.9780	0.9980	0.997
+0.0880 (+9.7%)				

At K=21, the system exhibits a clear precision–recall trade-off: MAP@21 remains virtually constant, while MAR@21 improves substantially.

7.3.5 Feature Importance Analysis

To understand learned feature interactions, we analyzed the HistGradientBoostingRegressor's feature importance scores for the configuration with user-criitc features (Stage 4).

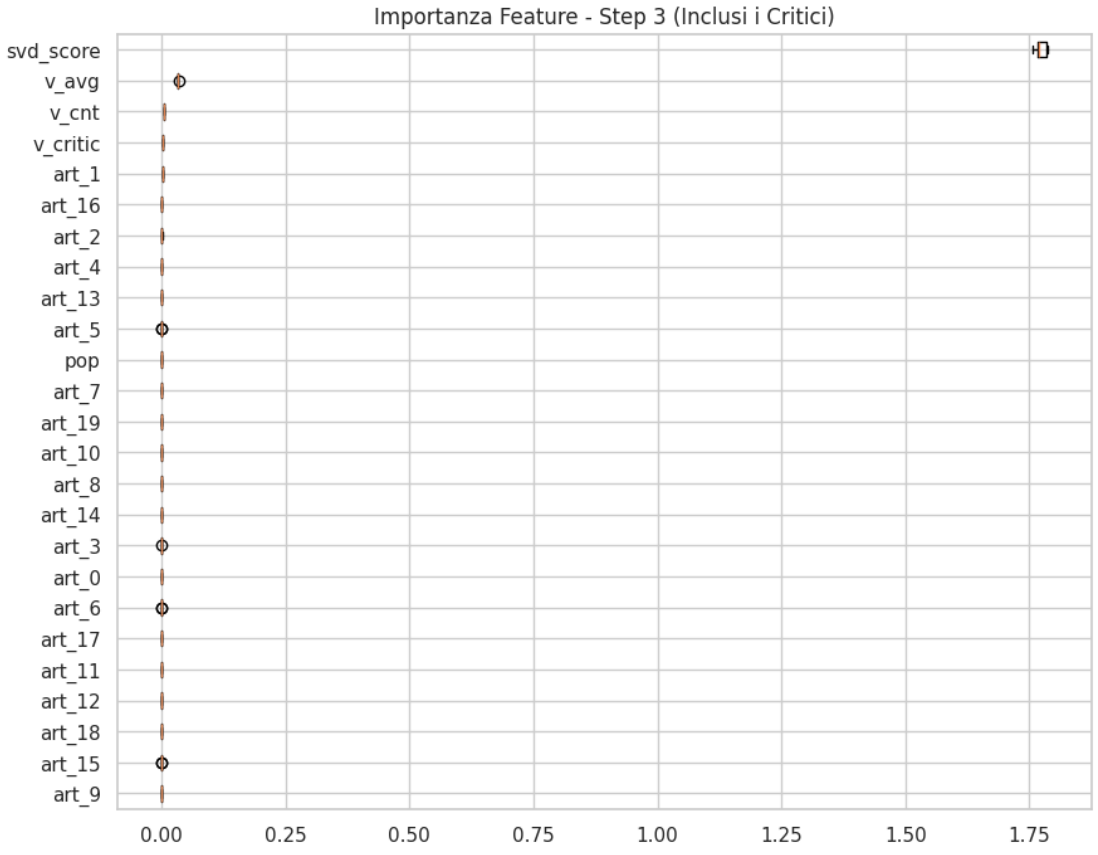


FIGURE 7.1: Feature importance for the final ablation configuration.

7.4 Discussion

7.4.1 Precision–Recall Trade-off

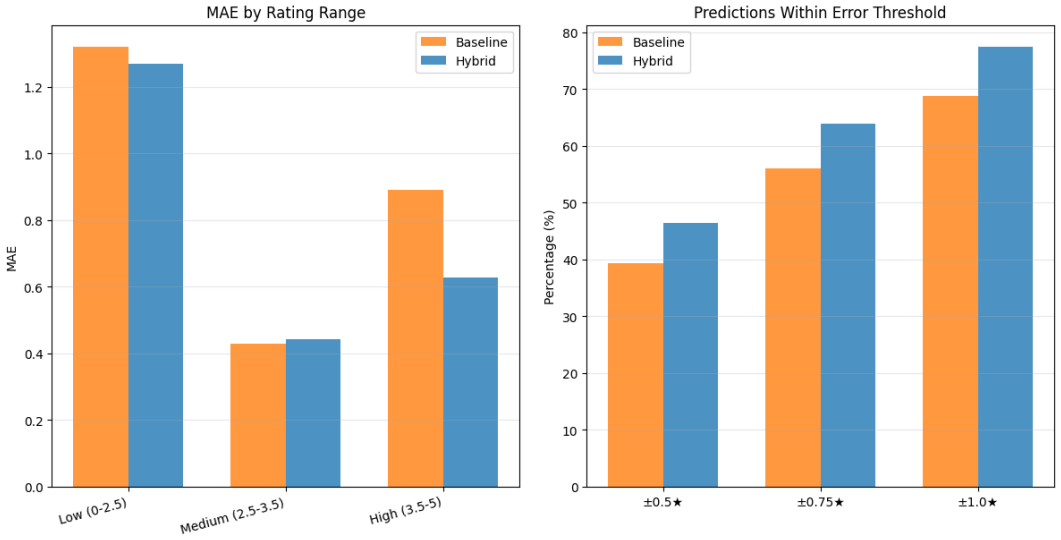


FIGURE 7.2: Comparitions between Baseline SVD++ and Hybrid

The experimental results reveal a tension between recommendation precision and recall: SVD++ performs strongly in terms of precision at small K, while the hybrid system significantly improves recall and coverage.

7.4.2 The User Feature Paradox

The negative impact of user engagement features on MAP@K contrasts with their substantial recall benefits. Possible explanations include popularity bias and complex feature interactions.

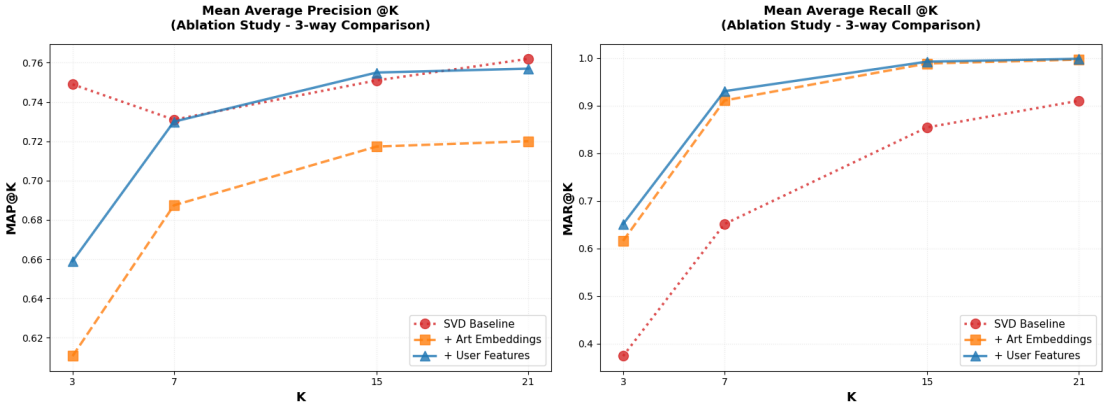


FIGURE 7.3: MAP and MAR Comparitions stage 3

7.4.3 Critic Features: Contribution

The absolute performance curves provide complementary insights into model behavior:

- **Precision analysis.** The SVD baseline exhibits competitive precision at $K = 3$ (MAP@K ≈ 0.75), suggesting strong performance for the very top recommendations based purely on collaborative filtering. However, this advantage diminishes as K increases, with the gap narrowing by $K = 21$. In contrast, the art-embeddings model shows substantially lower precision at $K = 3$ (MAP@K ≈ 0.61), confirming the detrimental effect observed in the incremental analysis. At larger cutoffs, both the user-features model and the complete model (including critic features) converge to similar precision values (MAP@K ≈ 0.76–0.77 at $K = 21$), suggesting diminishing returns from feature engineering when the recommendation list becomes longer.

- **Recall analysis.** The recall curves show the opposite trend: the SVD baseline has poor recall at $K = 3$ (MAR@K ≈ 0.38) but improves substantially by $K = 21$ (MAR@K ≈ 0.91). This indicates that collaborative filtering excels at precision but may suffer from limited coverage in short lists. Models incorporating content and user features achieve higher recall already at $K = 3$ (MAR@K ≈ 0.61–0.66), demonstrating a stronger ability to retrieve relevant items early in the ranking. The complete model with critic features reaches near-perfect recall (MAR@K ≈ 0.99–1.00) at $K = 15$ and $K = 21$, suggesting that the multi-faceted feature set provides broad coverage of user preferences.

- **Precision–recall trade-off.** Overall, the results highlight an architectural trade-off: the SVD baseline prioritizes precision over recall, whereas hybrid models achieve a better balance. The art-embeddings model performs poorly on both metrics at $K = 3$, reinforcing that content features require integration with behavioral signals to be effective. The convergence of all models at higher K values suggests that the most meaningful differences emerge in ranking quality for small top-K recommendation lists.

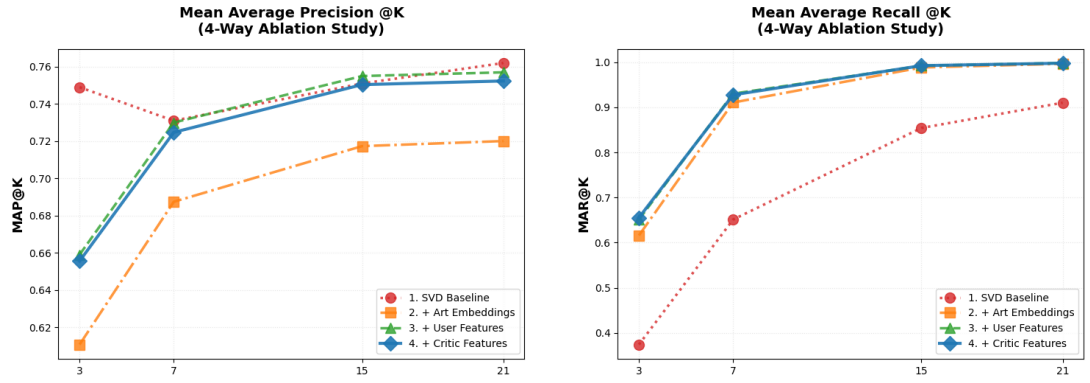


FIGURE 7.4: MAP and MAR Comparitions stage 4 critic features

The incremental analysis reveals several critical insights into the contribution of each component to the overall system performance:

- **Step 1 (+ Art Embeddings).** The incorporation of content-based art embeddings produces a negative impact on MAP@K across all tested cutoffs ($K \in \{3, 7, 15, 21\}$), with decrements ranging from approximately -0.038 to -0.140. This counterintuitive result suggests that adding content features without proper calibration introduces noise into the collaborative baseline. The most pronounced degradation occurs at $K = 3$ (ΔMAP@K ≈ -0.140), indicating that content features particularly harm the very top-ranked recommendations when used in isolation from behavioral signals.

- **Step 2 (+ User Features).** Adding user demographic and behavioral features yields consistent improvements across all tested cutoffs, with gains ranging from approximately $+0.037$ to $+0.048$. This recuperative effect partially compensates for the losses observed in Step 1, suggesting that user-specific contextual information helps the model exploit content embeddings more effectively. The relatively uniform improvement across different K values indicates that user features provide stable discriminative power regardless of recommendation list length.

- **Step 3 (+ Critic Features).** Incorporating professional critic ratings and reviews results in marginal negative effects (ΔMAP@K ≈ -0.002 to -0.004). This indicates that expert assessments provide limited additional value once collaborative and content signals are already combined. A possible explanation is that crowd-sourced preferences may diverge from critical consensus, particularly in domains where subjective taste dominates.

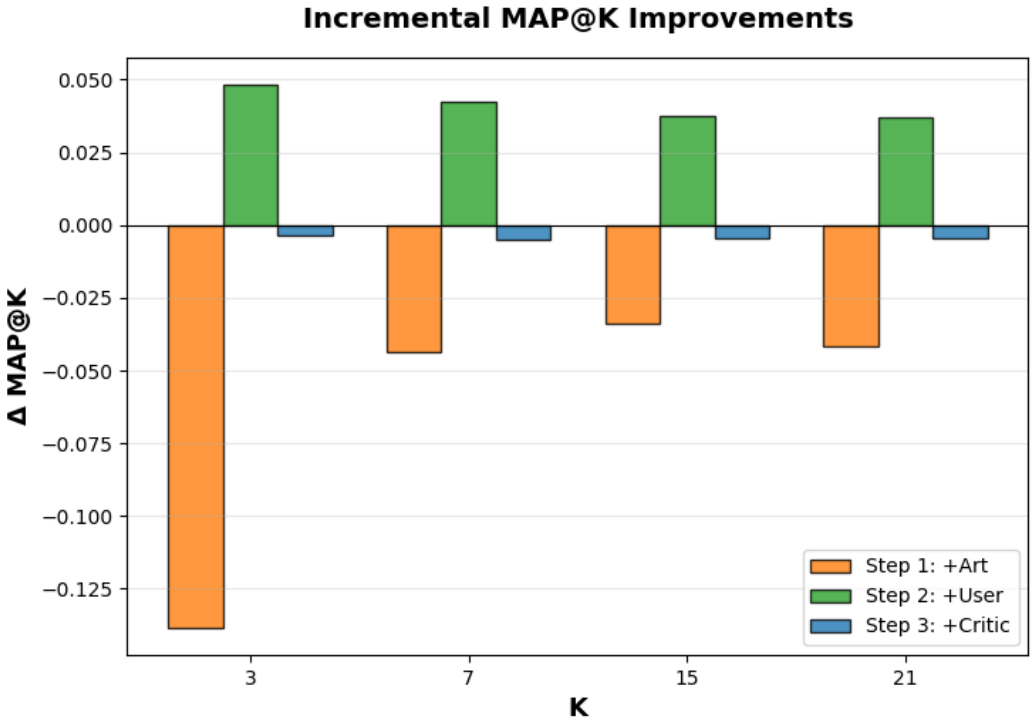


FIGURE 7.5: MAP and MAR Delta Contributions

7.5 Limitations and Future Work

7.5.1 Current Limitations

- **Fixed threshold**: the relevance threshold (rating ≥ 2.0) is arbitrary.
- **Limited hyperparameter tuning**: the HistGradientBoostingRegressor uses fixed hyperparameters.
- **Static feature representation**: artistic embeddings are pre-computed and fixed.
- **Single evaluation perspective**: diversity, novelty, and serendipity are not measured.

7.5.2 Future Research Directions

1. **Multi-objective optimization**: explicitly balance precision and recall.
2. **Attention-based fusion**: replace gradient boosting with neural attention mechanisms.
3. **Contextual features**: incorporate temporal signals and session context.
4. **Explainability analysis**: investigate which artistic dimensions influence recommendations.
5. **Large-scale evaluation**: validate findings on larger datasets.

Chapter 8

Conclusions

The ablation study provides a nuanced view of how each feature family contributes to the performance of a hybrid recommender system. A key (and somewhat counterintuitive) result emerges already in **Step 1**: when introduced in isolation, content embeddings can *decrease* recommendation quality. This suggests that content-based signals do not automatically translate into better personalization and, if not properly calibrated, may inject noise or bias into the collaborative baseline. A plausible explanation is a misalignment between *content similarity* and *preference similarity*: movies that are close in theme or style are not necessarily appealing to the same users.

The gains observed in **Step 2** highlight the importance of richer user context. User-level features (demographics, interaction patterns, and behavioral indicators) provide an additional layer of personalization that helps reconcile the gap between content-based representations and individual taste. In other words, effective hybrid systems should not merely concatenate heterogeneous features, but rather exploit user information to *weight* and *interpret* content signals in a user-dependent way.

In **Step 3**, critic-related features contribute only marginally. Although professional reviews offer high-quality domain expertise, their limited impact suggests that, in this setting, aggregate user behavior captures preference diversity more effectively than expert consensus. This observation is consistent with the well-known *wisdom of crowds* effect in collaborative filtering, and it also hints that critic signals may be more valuable in cold-start regimes or for niche items where user feedback is sparse.

Finally, the precision–recall analysis clarifies the complementary strengths of the investigated approaches. Pure collaborative filtering (SVD++) tends to excel in precision, but it is less effective in recall, especially for small K. This behavior is compatible with the popularity bias often observed in matrix factorization methods, which favors globally popular items and may overlook a broader range of relevant content. By injecting controlled diversity through side information, the hybrid models alleviate this limitation and achieve substantially better recall while preserving competitive precision.