Classical Recommender Algorithms & Offline Evaluation

Anton Yurchenko Volodymyr Leha Dariia Yurko

February 2026

1 Introduction

In this work, we explore how different classical recommender algorithms perform on the industry-standard dataset. Our goal is to build a practical implementation as well as provide the theoretical foundations for collaborative and content-based recommender algorithms as well as define a solid and universal evaluation framework to compare their performance.

2 EDA

The MovieLens dataset used in this study contains 1 million user ratings for movies along with user and movie identifiers. We decided to explore this dataset to reveal key characteristics and challenges that may influence the performance of recommendation algorithms.

2.1 Data review

First thing to analyze was what data we are dealing with. for this we built a simple distribution histogram visualizing the user-movie rating dependency.

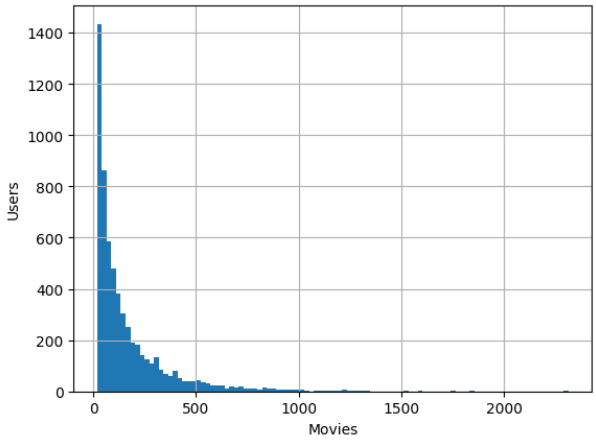


Figure 1: User-movie ratings histogram.

An important finding was that all users in the dataset have at least 20 reviews, so there are roughly no cold-start users.

A sparsity heatmap was created to visualize the structure of the user-item rating matrix. In this plot, dark areas represent missing ratings and light areas represent available ratings. This visualization highlights the high level of sparsity in the dataset, indicating that many users have rated only a few movies. Sparsity is crucial for matrix completion tasks since it affects both the accuracy and the complexity of the prediction models. The sparsity of the entire dataset is 95.53%. This is a usual level of sparsity for real-world dataset. It motivates the use of matrix factorization algorithms, while still leaving enough meaningful information to learn the structure.

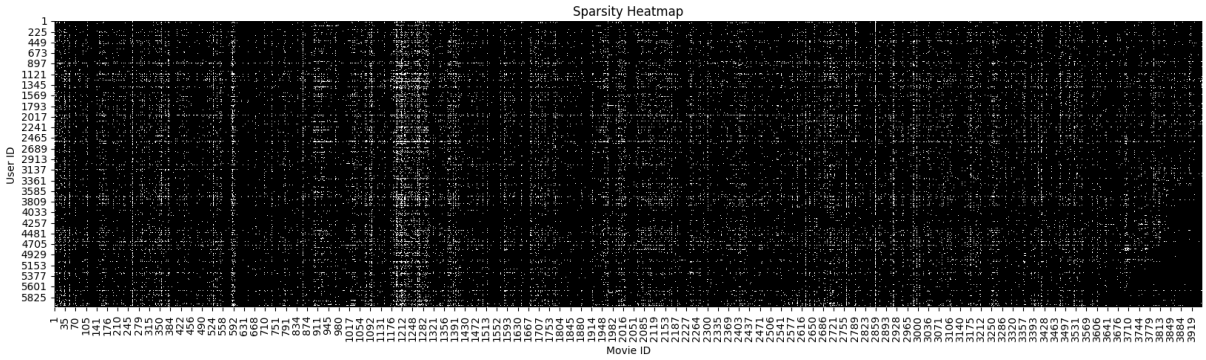


Figure 2: Sparsity heatmap of user-movie ratings.

Another quick observation is the rating distribution. We can assume the ratings are somewhat normally distributed around the mean of 4, but there are not enough categories to give a good estimation. What we can see from this plot is that the most popular classes are 3, 4, and 5, while 1 and 2 are somewhat underrepresented.

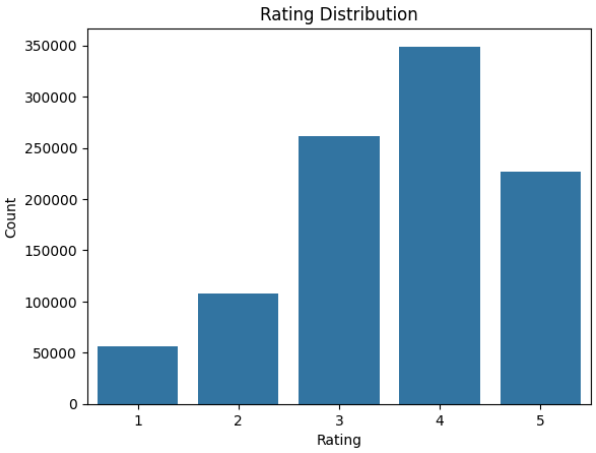
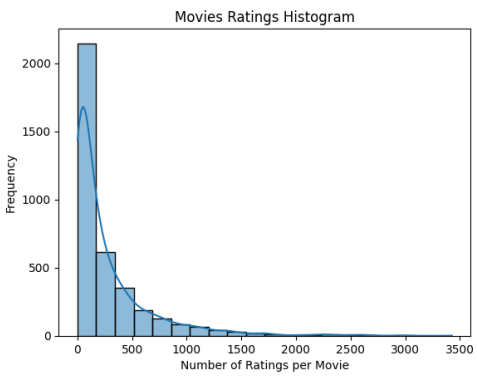
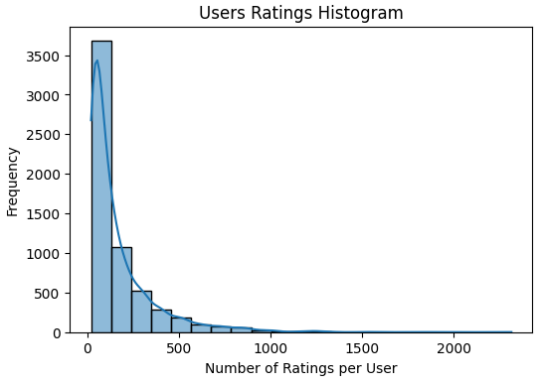


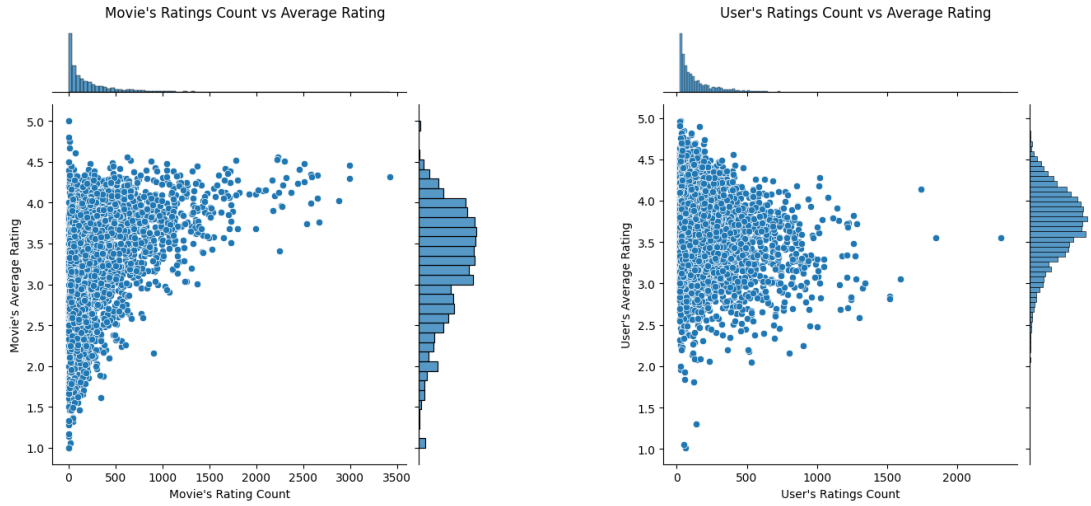
Figure 3: Distribution of user ratings

Now we come to the interesting part. The distribution of moview and user ratings. TBoth distributions are highly skewed, meaning that most of the movies ahve a small number of ratings, while a few have insanely large number of reviews. At the same time similar thing happens with users: few users leave a lot of ratings while most of them are inactive. This is actually where the high sparsity comes from.





We can further investigate this skewness by exploring the distribution of user and movie ratings.



From this information we can see that almost a third of the users (28.86%) left less than 50 reviews, and more than half of them (51.24%) left less than 100. All of this keeping in mind that dataset is filtered so that there are no purely cold-start users.

Another important thing to look at is the temporal user activity. This shoes a very interesting and weird picture.

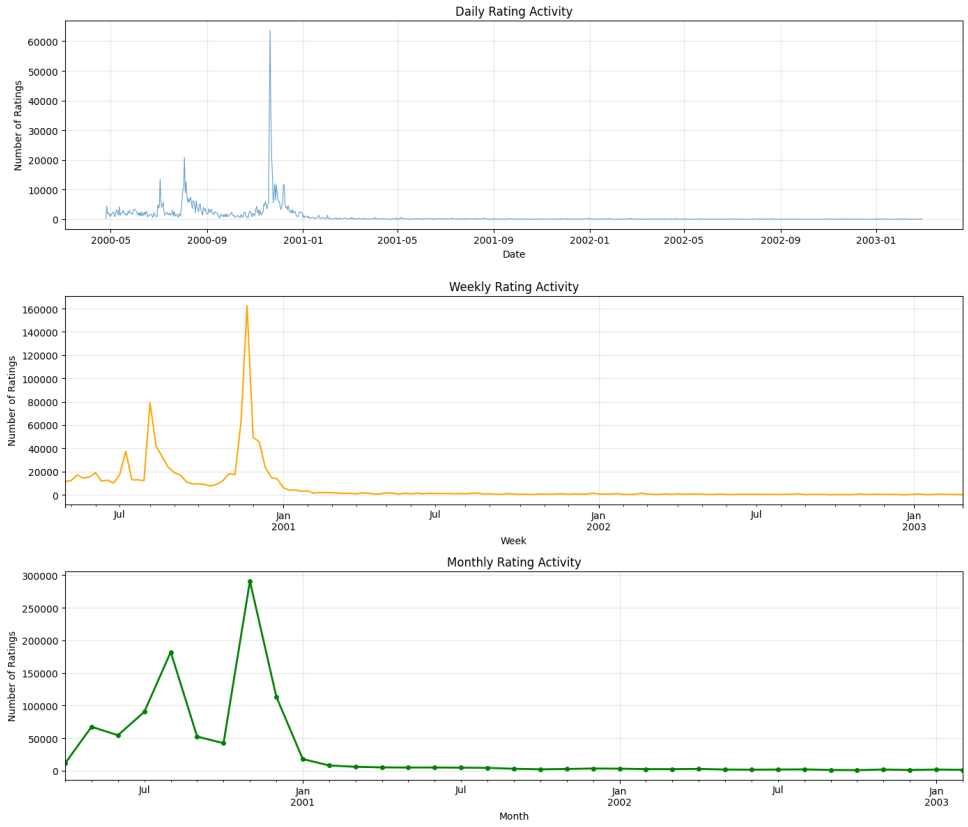


Figure 6: Temporal user activity

What is interesting about this graph is that it shows that there were weird spikes of activity early in the observation period, while in the latter more than two thirds the user activity was surprisingly stable.

2.2 Data pathologies

1. First things first, in any dataset an important thing is the distribution of data. In recommender systems this is often for the data to be skewed. In our case, both user ratings and movie ratings

have very skewed distributions. Once again, this is the causes (or products) of a highly sparse datasets. And these are the exact issues we are trying to mitigate while building recommender systems. However, high level of skewness imposes difficulties during training, and at the same time motivates the use of matrix factorization methods, since they try to fill in the missing gaps in the dataset.

2. Another important thing to talk about are cold-start users. We have a filtered dataset so that none of the users are truly cold start, but for both collaborative and content-based algorithms it is important so that users have as many features as possible, and even the dataset being filtered might not mitigate the damage to training.

3. Lastly, the temporal distribution is clearly degenerate in this dataset, which in turn makes it hard to capture seasonal tendencies for users.

3 Evaluation strategy

This is to describe the offline evaluation methodology design.

3.1 Data Split Strategy

Random splits are not used, as they may introduce temporal leakage by allowing future interactions to influence model training. Instead, temporal splitting strategy is applied to simulate real-world recommendation scenarios.

All interactions are sorted by timestamp. A global cutoff is defined at the quantile of the timestamp distribution. Interactions occurring at or before the cutoff form the training set, while interactions after the cutoff form the test set. This setup reflects a realistic deployment scenario where models are trained on historical data and evaluated on future behavior.

3.2 Train / Validation / Test Mechanics

- **Training set**: Interactions prior to the temporal cutoff.
- **Validation set** (optional): Recent interactions from the training period used for hyperparameter tuning.
- **Test set**: Interactions occurring after the cutoff. As the dataset is large enough (more than 1M rows), 10-20% of the dataset was assigned to the test set.

This structure ensures that evaluation is conducted exclusively on unseen future interactions.

3.3 Evaluation Objectives

The evaluation addresses two complementary tasks:

- Rating prediction, measuring how accurately explicit ratings are estimated;
- Top-K ranking evaluation, focusing on the quality of recommended item lists.

3.4 Metrics

3.4.1 Rating Prediction Metrics:

RMSE (Primary): Measures the magnitude of rating prediction errors, emphasizing large deviations.

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

MAPE (Secondary): Measures relative prediction error, providing interpretable percentage-based diagnostics.

$$\text{MAPE} = \frac{100}{N}\sum_{i=1}^{N}\left|\frac{r_i - \hat{r}_i}{r_i}\right|$$

3.4.2 Ranking Metrics:

Precision@K (Additionally): Proportion of relevant items in the top-K recommendations. Items with ratings ≥ 4 in the test set are considered relevant.

$$\text{Precision@K} = \frac{|\text{Rec}_K(u) \cap \text{Rel}(u)|}{K}$$

3.5 Deliverables

- Evaluation code module implementing RMSE, MAPE, Precision@K;
- Evaluation methodology design document, describing the data splits, metrics, and evaluation.

3.6 The Evaluation Coverage and Limitations

Captured:

- Realistic future prediction via strict global temporal split (no leakage);
- Rating accuracy on explicit feedback (RMSE, MAPE);
- Top-K recommendation relevance (Precision@K);
- Consistent comparison across different recommendation approaches.

Not captured:

- Online evaluation;
- Cold-start users and items;
- User-level temporal fairness (sparse histories);
- Position bias, diversity and novelty.

4 Matrix factorization

4.1 ALS

Alternating Least Squares models the user–item matrix $\mathbf{M}$ as the product of two low-rank matrices:

$$\hat{\mathbf{M}} = \mathbf{U}\,\mathbf{V}^T,$$

where:

1. $\mathbf{U}$ is an $m \times r$ matrix representing r-dimensional latent features for each of the m users,
2. $\mathbf{V}$ is an $n \times r$ matrix representing r-dimensional latent features for each of the n items.

4.1.1 In-Depth Analysis

Alternating Least Squares is a collaborative filtering algorithm used in recommendation systems to predict user preferences. The goal of ALS is to decompose the user-item matrix into two smaller matrices, U and V, which capture latent features of users and items. By alternating between fixing U and optimising V, and vice versa, ALS minimizes the least squares error to make accurate predictions.

We have our original matrix R of size $u \times i$ with our users and items. We then want to find a way to turn that into one matrix with users and hidden features of size $u \times f$ and one with items and hidden features of size $f \times i$. In U and V we have weights for how each user/item relates to each feature. What we do is we calculate U and V so that their product approximates R as closely as possible: $R \approx U \times V$.

By randomly assigning the values in U and V and using least squares iteratively we can arrive at what weights yield the best approximation of R. The least squares approach in it's basic forms means fitting some line to the data, measuring the sum of squared distances from all points to the line and trying to get an optimal fit by minimising this value. With the alternating least squares approach we use the same

idea but iteratively alternate between optimizing U and fixing V and vice versa. We do this for each iteration to arrive closer to $R = U \times V$.

The solution is to merge the preference (p) for an item with the confidence (c) we have for that preference. We start out with missing values as a zero preference with a low confidence value and existing values a positive preference but with a high confidence value.

We set the preference (p)

$$p_{ui} = \begin{cases} 0, & r_{ui} > 0 \\ 0, & r_{ui} = 0 \end{cases}$$

Basically our preference is a binary representation of our feedback data r. If the feedback is greater than zero we set it to 1. Make sense.

The confidence (c) is calculated as follows:

$$c_{ui} = 1 + \alpha r_{ui}$$

Here the confidence is calculated using the magnitude of r (the feedback data) giving us a larger confidence the more times a user has played, viewed or clicked an item. The rate of which our confidence increases is set through a linear scaling factor α. We also add 1 so we have a minimal confidence even if $\alpha \times r = 0$.

The goal now is to find the vector for each user (xu) and item (yi) in feature dimensions which means we want to minimize the following loss function:

$$\min_{y_*, y_*} \sum_{u,i} c_{ui}(p_{ui} - x_u^\top y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

If we fix the user factors or item factors we can calculate a global minimum. The derivative of the above equation gets us the following equation for minimizing the loss of our users:

$$x_u = \left(Y^\top C_u Y + \lambda I\right)^{-1} Y^\top C_u p(u)$$

And the this for minimizing it for our items:

$$y_u = \left(X^\top C_i X + \lambda I\right)^{-1} X^\top C_i p(i)$$

One more step is that by realizing that the product of Y-transpose, Cu and Y can be broken out as shown below:

$$Y^\top C_u Y = Y^\top Y + Y^\top \left(C_u - I\right) Y$$

Now we have $Y^\top Y$ and $X^\top X$ independent of u and i which means we can precompute it and make the calculation much less intensive.

$$x_u = \left(Y^\top Y + Y^\top \left(C_u - I\right) Y + \lambda I\right)^{-1} Y^\top C_u p(u)$$

$$y_u = \left(X^\top X + X^\top \left(C_i - I\right) X + \lambda I\right)^{-1} X^\top C_i p(i)$$

To make recommendations we calculate the dot product between our user vectors and the transpose of our item vectors. This gives us a recommendation score for our each user and each item:

$$score = U \cdot V^\top$$

4.1.2 Advantages

1. Each user (or item) update can be computed independently, making ALS well-suited for distributed systems.
2. Each iteration involves solving parallelizable least squares problems, which can handle large-scale data.
3. Built-in regularization (λ) mitigates overfitting and addresses noisy data.

4.1.3 Disadvantages

1. The choice of latent dimension r can strongly impact performance and generalization.
2. New users or items with no prior data remain problematic.
3. Tuning the number of iterations and λ is essential for stable and accurate results.

4.1.4 Experiments

The experiments were mainly focused on finding the best hyperparameters for the model. The tests were performed on the number of latent factors as well as the α confidence scaling parameter. The optimization metric was traditionally RMSE.

Factors	Alpha	Validation RMSE
10	20	3.6193
10	60	3.5392
40	20	3.5752
40	60	**3.4497**

Table 1: Validation RMSE for different combinations of ALS hyperparameters

The error fluctuated insignificantly. Still for the best model with the number of factors of 40 and $\alpha = 60$, and with increased number of epochs, the resulting evaluation metrics were as follows:

RMSE	MAPE (%)	Precision@5	Precision@10	Precision@2
3.1120	81.38	0.0684	0.0619	0.0588

Table 2: Validation RMSE for different combinations of ALS hyperparameters

4.2 FunkSVD

FunkSVD is a matrix factorization algorithm designed for collaborative filtering with explicit feedback. Similar to ALS, it approximates the user–item rating matrix as a product of two low-rank matrices:

$$\hat{\mathbf{R}} = \mathbf{U}\,\mathbf{V}^T,$$

where:

1. $\mathbf{U} \in \mathbb{R}^{u \times k}$ represents k-dimensional latent feature vectors for users,
2. $\mathbf{V} \in \mathbb{R}^{i \times k}$ represents k-dimensional latent feature vectors for items.

Unlike ALS, FunkSVD does not solve closed-form least squares problems. Instead, it learns these latent representations through iterative optimization using stochastic gradient descent (SGD).

4.2.1 In-Depth Analysis

FunkSVD was originally introduced during the Netflix Prize competition and is designed to operate directly on sparse rating data without requiring imputation of missing values. The algorithm assumes that observed ratings arise from interactions between latent user preferences and latent item characteristics.

Given an observed rating r_{ui}, FunkSVD predicts it as:

$$\hat{r}_{ui} = x_u^\top y_i$$

where x_u is the latent feature vector for user u and y_i is the latent feature vector for item i.

The objective is to minimize the regularized squared error over observed ratings only:

$$\min_{x_*, y_*} \sum_{(u,i)\in\mathcal{K}} \left(r_{ui} - x_u^\top y_i\right)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right),$$

where $\mathcal{K}$ denotes the set of observed user–item interactions and λ controls regularization.

Unlike ALS, FunkSVD treats missing values as unknown rather than as zero-valued feedback with confidence weights. This makes it particularly suitable for explicit-feedback datasets such as movie ratings.

FunkSVD optimizes the objective using stochastic gradient descent. For each observed rating (u, i), the prediction error is computed as:

$$e_{ui} = r_{ui} - x_u^\top y_i.$$

The latent vectors are then updated according to:

$$x_u \leftarrow x_u + \eta \left(e_{ui} y_i - \lambda x_u \right),$$

$$y_i \leftarrow y_i + \eta \left(e_{ui} x_u - \lambda y_i \right),$$

where η is the learning rate.

These updates are applied iteratively for all observed ratings across multiple epochs. By shuffling observations each epoch, the algorithm reduces bias and improves convergence stability.

4.2.2 Advantages

1. Efficient for sparse explicit-feedback datasets, since it operates only on observed ratings.
2. Simple to implement and flexible with respect to loss functions and extensions.
3. Often converges quickly with properly tuned learning rate and regularization.

4.2.3 Disadvantages

1. Requires careful tuning of learning rate, regularization, and number of epochs.
2. Sequential SGD updates make parallelization more difficult than ALS.
3. Susceptible to local minima and unstable training if hyperparameters are poorly chosen.

4.2.4 Experiments

The experiments were mainly focused on finding the best hyperparameters for the model. The tests were performed on the number of dimensions in latent space k as well as the η learning rate. The optimization metric was traditionally RMSE.

k	η	**Validation RMSE**
10	0.003	0.9434
10	0.01	**0.9118**
40	0.003	0.9469
40	0.01	0.9218

Table 3: Validation RMSE for different combinations of FunkSVD hyperparameters

Once again, the error fluctuated insignificantly. Still for the best model with $k = 10$ and $\eta = 0.01$, and with increased number of epochs, the resulting evaluation metrics were as follows:

RMSE	**MAPE (%)**	**Precision@5**	**Precision@10**	**Precision@2**
0.9151	27.79	0.0637	0.0632	0.0633

Table 4: Validation RMSE for different combinations of SVD hyperparameters

5 Similarity-based recommenders

5.1 Content-Based Filtering

5.1.1 Performance Comparison: Cosine vs Euclidean Similarity

Performance across different neighbor counts:

Key Observations:

1. Euclidean distance consistently outperforms cosine similarity across all neighbor counts.
2. Performance improves with more neighbors (n=50 optimal for both methods).
3. Euclidean advantage is consistent ($\sim$1.3% improvement in RMSE).
4. Both methods show diminishing returns beyond n=20 neighbors.

Number of Neighbors (n)	Cosine RMSE	Euclidean RMSE	Difference
5	1.1479	1.1388	-0.0091
10	1.1079	1.0964	-0.0115
20	1.0902	1.0765	-0.0137
50	1.0825	1.0686	-0.0139

Table 5: Detailed RMSE Analysis by Number of Neighbors

5.1.2 Cosine Similarity

Cosine similarity measures how similar two vectors are by computing the angle between them, focusing on direction rather than magnitude. It is scale-invariant and produces values in $[-1, 1]$, typically $[0, 1]$ for TF-IDF features. This measure works well for sparse, high-dimensional data and offers an intuitive interpretation of similarity. It is robust to feature scaling, easy to interpret, and commonly used for text-based representations. It ignores magnitude differences, which may hide patterns where feature size is important. **Best RMSE:** 1.083 (n=50)

5.1.3 Euclidean Distance

Euclidean distance measures the absolute difference between feature vectors and is converted to similarity = 1/(1+distance). This measure is sensitive to feature magnitudes and produces values in $[0, 1]$ after conversion. It captures absolute differences in feature importance more directly than angle-based measures. It accounts for magnitude differences in TF-IDF weights and achieved slightly better performance ($\approx 1.3\%$ improvement). It is sensitive to feature scaling, less commonly used for text features, and requires an explicit distance-to-similarity conversion. **Best RMSE:** 1.069 (n=50)

5.1.4 Scenarios Where Content-Based Filtering Fails

The system has several limitations typical of content-based recommenders. It performs poorly in cold-start cases, as movies without genre metadata and users with few ratings cannot be handled effectively; this can be mitigated through popularity-based fallbacks, user prompts, and demographic information. Limited feature diversity causes identical recommendations for movies with the same genre profiles, which can be addressed by enriching features with additional metadata such as actors, directors, and plot keywords. The model may also overspecialize, reducing diversity and discovery, requiring diversity constraints or serendipity mechanisms. Its performance depends heavily on metadata quality, making regular validation and updates necessary. Sparse user profiles and changing user preferences over time further reduce accuracy, which can be alleviated using popularity-based fallbacks, temporal features, and periodic retraining.

5.1.5 Bias Analysis

The system exhibits relatively low popularity bias because recommendations are driven by content features rather than item popularity, allowing niche items to surface, though quality signals may be missing; this can be mitigated by incorporating popularity or quality indicators into the ranking. Activity bias is less pronounced than in collaborative filtering, but active users still receive better recommendations than inactive users, which can be addressed through popularity-based fallbacks and prompts for additional ratings. Temporal bias is also reduced since genres are generally stable over time, although shifts in genre popularity are not captured; including release year, recency boosts, and time-decay mechanisms helps compensate. Demographic bias may emerge from rating patterns and should be monitored, with fairness constraints applied when necessary. Recommendation diversity can be limited, creating filter bubbles and reducing discovery, which calls for diversity constraints and serendipity mechanisms. Finally, feature representation bias may arise because TF-IDF favors common genres; this can be reduced through frequency normalization and genre importance weighting.

5.1.6 Deployment Recommendations

Euclidean distance is recommended as the primary similarity function due to its 1.3% RMSE improvement and better Precision@K for $K \geq 10$, while cosine similarity remains a viable alternative when interpretability or robustness is prioritized. The best-performing setup uses Euclidean distance with $n = 50$, TF-IDF genre features, and weighted-average prediction (RMSE 1.069), with smaller neighborhood sizes

offering speed–accuracy trade-offs. Implementation should be phased, starting with this baseline and later adding diversity constraints, popularity awareness, richer features, and temporal signals. Edge cases can be handled using popularity-based fallbacks and genre-based similarity, while system performance should be monitored through accuracy, ranking, engagement, diversity, fairness, and cold-start metrics, with alerts for significant degradation.

5.1.7 Next Steps and Future Considerations

Immediate improvements should focus on richer feature engineering by extending beyond genres to include metadata such as actors, directors, release year, plot keywords, temporal signals, and implicit feedback, alongside systematic hyperparameter tuning and explicit diversity enhancements. More advanced techniques include incorporating textual, visual, and extended metadata features, exploring alternative or learned similarity measures, and improving explainability through clearer feature-based recommendations. Fairness and ethical considerations require ongoing bias mitigation, transparent recommendation logic, and strong privacy practices with minimal data collection and user control. Scalability can be addressed through efficient computation using sparse representations, approximate nearest neighbor search, caching, and precomputed similarities to support real-time use. Finally, a robust evaluation framework should combine offline validation with online experimentation and long-term impact metrics such as retention, discovery, and user satisfaction.

5.2 Similarity-Based Collaborative Filtering

5.2.1 Similarity Functions Comparison

Both cosine similarity and similarity based on Pearson correlation are widely used in recommendation systems because they measure how similar users or items are, which is important for predicting preferences. They are used because they solve two specific problems: magnitude differences (Cosine) and user bias (Pearson).

As we already wrote, cosine similarity measures the angle between two vectors (users or items) in a multi-dimensional space. Cosine similarity captures directional similarity: users who rate items similarly in pattern. It works well when ratings are sparse (that is our case with sparsity 95%), because the cosine is calculated only on co-rated items.

Pearson similarity is very similar to cosine similarity, but it measures linear correlation between two vectors, i.e., whether their ratings increase and decrease together. Unlike cosine, it removes the mean of each user / item first. F.e., one user leaves high ratings (mostly 4–5 stars), another - stricter ratings (mostly 2–3 stars), Pearson still detects similarity in patterns.

5.2.2 Item-Based vs User-Based Approach Comparison

We decided to compare both item-item and user-user approaches, as they solve different problems. While user-based relies on social patterns (finding similar users), item-based relies on product associations (finding similar items).

The idea behind the user-based approach is to find users who are similar to the target user (neighbors) based on their ratings, and recommend items that similar users liked but the target user hasn't seen yet.

The idea behind the item-based approach is to compare items based on how similar they are according to user ratings, instead of comparing users. Then, recommend items similar to what the target user already liked.

5.2.3 Similarity-Based CF Experiments Results

For the experiments with similarity-based collaborative filtering approaches we developed 4 models:

- Item-based approach based on cosine similarity function;
- User-based approach based on cosine similarity function;
- Item-based approach based on Pearson similarity function;
- User-based approach based on Pearson similarity function.

Model	RMSE	MAPE (%)	Precision@10
Cosine Similarity (Item-based)	1.049	35.17	0.069
Cosine Similarity (User-based)	**0.975**	30.23	**0.130**
Pearson Similarity (Item-based)	2.818	32.69	0.081
Pearson Similarity (User-based)	**0.975**	**29.69**	0.083

Table 6: Similarity-Based CF Models Evaluation

The evaluation metrics results of the 4 experiments are in the table below.

Both cos_sim_user_based and pearson_sim_user_based models performed significantly better than item-based approaches in predicting the exact rating values. RMSE is nearly identical (≈ 0.975), which is a solid score for MovieLens 1-5 rating datasets. In MAPE pearson_sim_user_based has a slight edge (29.69%) over cosine (30.23%), meaning it makes slightly smaller percentage errors on average. pearson_sim_item_based model failed significantly with an RMSE of 2.81. Considering rating scale 1-5, an error of 2.81 is huge. This suggests the model is failing to find correlated items or is highly unstable (maybe, due to sparse data where items have very few common users).

At the same time, precision@k is more critical metric recommendation systems. cos_sim_user_based model achieved 0.13, which is 56% higher than the next best model (pearson_sim_user_based at 0.083).

So, from 4 experiments, we can conclude that cosine similarity cos_sim_user_based won over other 3 models both on rating prediction metrics and ranking metrics.

Also, we see that for the MovieLens dataset data user-based approach wins over item-based. At the same time cosine similarity approach show better performance on rating prediction metrics, while pearson similarity models have higher ranking metrics.

5.2.4 Advantages and Disadvantages of Similarity-Based CF Approach

Advantages:

- Simplicity and interpretability, as we can easily explain why a recommendation was made.
- Domain agnostic, no item metadata or user data required.
- Serendipity, as the system can recommend items a user has no history of interacting with.

Disadvantages:

- The cold start problem. If a user has rated nothing, their similarity vector is empty. The system cannot recommend anything. If a new movie is added but no one has rated it yet, it will never be recommended.
- Sparsity problem, because to calculate similarity, we need overlaps.
- Scalability, as to find similar users, we compare the active user against all other users. If we have 10M users, calculating real-time user-based similarity becomes too slow for a real-time inference.
- Some user have unique taste. Collaborative filtering fails for them completely because there is no "neighbors" to get recommendations from.

6 Findings & conclusion

6.1 Resulting metrics comparison

Model	RMSE	MAPE (%)	Precision@10
ALS	3.1120	81.38	0.0619
FunkSVD	**0.9151**	**27.79**	0.0632
Cosine Item-based	1.049	35.17	0.069
Cosine User-based	0.975	30.23	**0.130**
Pearson Item-based	2.818	32.69	0.081
Cosine User-based	0.975	29.69	0.083

Table 7: Comparison table for metrics across different recommender system algorithms.

As can be seen from the comparison table, interestingly, the more classic approach like FunkSVD gave better results than more sophisticated similarity-based algorithms. Precision is comparable among all the methods (taking cosine item-based as a representative of the content based methods).

6.2 Summary

This work implemented and evaluated six recommender models across two broad paradigms — matrix factorization (ALS, FunkSVD) and similarity-based methods (content-based and collaborative filtering) on the MovieLens 1M dataset using a strict temporal evaluation split.

6.2.1 Performance comparison

The results present a somewhat counterintuitive ranking. FunkSVD achieved the strongest overall profile: the lowest RMSE (0.9151), the lowest MAPE (27.79%), and competitive Precision@10 (0.0632). Cosine user-based collaborative filtering matched FunkSVD on RMSE (0.975) and significantly outperformed every other model on Precision@10 (0.130), which is the metric that most directly reflects the quality of actual recommendation lists a user would see.

ALS is the clear underperformer. Its RMSE of 3.112 on a 1–5 scale is effectively uninformative, and its MAPE of 81.38% confirms that the model's point predictions are nearly meaningless.

Pearson item-based CF is the other notable failure, with an RMSE of 2.818. The likely cause is sparsity compounded by the item-based framing: to compute Pearson correlation between two items, you need a meaningful set of users who rated both. In a dataset with 95.53% sparsity, many item pairs share very few common raters, making the correlation estimates extremely noisy.

Content-based filtering (Euclidean, n=50) achieved an RMSE of 1.069, placing it in the middle of the pack. This is a reasonable result given that it operates entirely on genre metadata, which is a very low-dimensional signal. It is not surprising that it cannot match methods that leverage the full structure of the rating matrix.

6.2.2 Failure scenarios

- Each model class fails in predictable and distinct ways. Matrix factorization methods (ALS, FunkSVD) degrade sharply for users and items at the tail of the activity distribution.
- Similarity-based collaborative filtering fails most visibly for users with unique tastes. If a user's rating pattern does not meaningfully correlate with any other user's pattern, the neighbor-based prediction degrades toward the global mean, which is uninformative.
- Content-based filtering fails when the feature space is too narrow. Genre alone cannot distinguish between, say, two action comedies with very different tones, casts, and audience appeal.

6.2.3 Bias Analysis

The models performance is significantly influenced by several data pathologies and biases. Here is a breakdown of how these biases affect the results:

Popularity & Skew Bias. The interaction distribution is highly skewed. A small fraction of more rated movies dominate the data.

Activity Bias: The report notes that active users receive better recommendations because they have richer history vectors. The 51% of users with ≤ 100 ratings will see lower quality results than the power users.

Temporal Bias: User activity showed "weird spikes" early in the dataset before stabilizing. Since the evaluation used a global temporal split, models trained on the "spiky" historical data might struggle to generalize to the "stable" future test set if time-decay isn't applied.

Sparsity: At 95.53% sparsity, some models collapsed because the probability of two users rating the exact same pair of items is too low to form a reliable link.

6.2.4 Deployment Recommendation

If forced to choose a single model, cosine user-based collaborative filtering is the best option. Its top Precision10 (0.130) delivers the strongest ranking performance, which directly impacts user experience. However, relying on it alone is risky due to RMSE limitations and cold-start issues. A practical deployment should be an ensemble: cosine user-based CF for users with sufficient history, content-based

filtering for new or sparse users, and FunkSVD to calibrate confidence or resolve ties. Euclidean distance is recommended as the primary similarity function due to its 1.3% RMSE improvement and better Precision@K for $K \geq 10$. ALS is avoided here, as its implicit-feedback design suits click-stream data, not explicit ratings.