Ranking Oriented, Hybrid, and Online-Aware Recommender Systems

February 23, 2026

1 Introduction

In this part we move beyond point-wise prediction and classical collaborative filtering into ranking-oriented, hybrid, and online-aware recommender systems. Our goal is to provide a practical implementation, experiments and theoretical background for a range of advanced recommender system-related topics.

2 Ranking Heuristics & Graph-Based Signals

For this section we implemented a couple of simple, non-learned ranking heuristics that are commonly used in recommender systems as baseline or fallback mechanisms. The reason for having these heuristics is to be able to provide recommendation when there is not enough personalized data to build a sophisticated model.

2.1 Popularity-based ranking

Popularity-based ranking orders items by the number of interactions they receive from users. We can then weight this number by the average rating of a specific item, we get a latent variable that encodes the relationship between the engagement and satisfaction from this movie. The assumption here is that the more interactions a movie has, the more appealing it is for a broader audience. This is the same assumption people make when deciding to buy a bestseller book, for example.

This way of ranking is often used as a baseline method, since it easily launches with a lot of cold-start items. However, this produces an unwanted symptom that popular items continue to remain popular and receive a lot of new interactions, while long-tail items fade into the unknown. This harms fairness.

2.2 Recency-based ranking

Recency-based ranking sorts items by the timestamp of their most recent interaction. The assumption here is that the items with most recent interactions are more relevant and popular in the given time frame. This way the system boosts the ratings of fresher content.

This type of recommender systems is common in environments where the content is very short-lived, for example, news feeds. However, they are not suitable for environments where there are items that can be considered to be ”classic”, for example, literature. Since it will bury the old but gold books.

2.3 Page Rank

Page rank is a graph-based ranking algorithm originally developed as a mechanism behind Google. It treats the collection of items (or pages) as a directed graph, assigning each node a score proportional to the scores of its incoming neighbors. In web search, a page that is linked to by many high-ranked pages receives a high rank. In our implementation, we build an item–item co-occurrence graph: each movie is a node, and we draw an edge between two movies if they are watched by the same user.

By propagating scores through the network, PageRank identifies items that sit at the “center” of the item graph. Variability in its application across domains shows the ability to produce stable rankings even in noisy networks. However, page rank is a static algorithm, thus it does not account for the age or recency of nodes. Studies on growing networks show that realistic temporal effects make PageRank fail in individuating the most valuable nodes. When the temporal decay of relevance and activity differ, the redistribution of PageRank scores becomes biased toward old or very recent nodes, and simpler measures can outperform it.

3 Hybrid Recommender Systems

For this section we implemented a hybrid recommender system that blends collaborative and heuristic signals. Heuristic scores, which are popularity, recency or page rank, are combined with collaborative scores after min-max normalization.

3.1 System Elements

Collaborative filtering captures patterns in user-item-interactions. By comparing users’ rating vectors, it infers preferences from similar users’ behaviour. The assumption is that items liked by similar users are relevant.

Heuristic signals are global in turn. Popularity assumes that widely interacted items are good defaults; recency - that recently interacted items are trending; and page rank spreads influence through co-occurrence graphs.

A hybrid recommender combines the strengths of these signals. Weighted hybridization is one of the simplest techniques: independent models produce scores which are normalized to a common range and then combined by a convex combination.

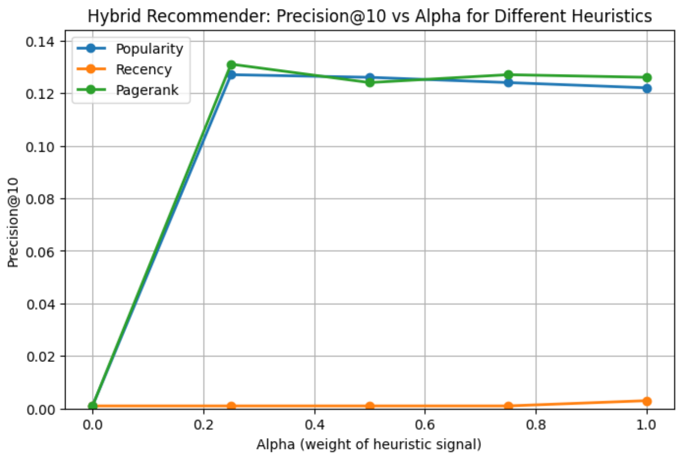
3.2 Implementation

As mentioned above, the implemented hybrid uses item-based collaborative filtering for the CF signal. A user-item rating matrix is constructed, cosine similarity is computed between item vectors, and a user's preference for a candidate item is predicted as a weighted sum of the user's ratings on similar items. The heuristic signal is computed via heuristic ranking. Both signals are min–max normalized, and the final score is computed as:

$$hybrid\_score(u, i) = \alpha \cdot heuristic(i) + (1 - \alpha) \cdot collaborative\_score(u, i)$$

, where $\alpha \in [0, 1]$ is a weighting parameter that controls the influence of the heuristic ranking.

3.3 Experiments



On this graph we can see the results of the experiments. Here, we see that, firstly, collaborative filtering on its own produces very low scores, and they get largely improved by the heuristics ranking. Recency ranking does not affect the precision score, most likely because the temporal distribution is very degenerate (see EDA from the previous lab). Both page rank and popularity heuristics largely increase the precision, however, they do not differ much between each other, with the precision score being almost identical for both page rank and popularity heuristics throughout all the non-zero α values.

4 Learning-to-Rank with Pairwise Optimization

To explore pairwise Learning-to-Rank approaches, we implemented the Bayesian Personalized Ranking (BPR-Opt) model. The core of the BPR-Opt approach is the transition from the absolute ratings to relative rankings, it shifts from answering the question "What rank will user A give to movie B?" to "Does user A prefer movie B over movie C?". The fundamental data unit in BPR-Opt is not a single interaction, but a triplet:

- user;
- positive item (something the user liked/interacted with);
- negative item (something the user ignored/have never seen).

In general, the BPR-Opt model is applied to problems with implicit feedback. In the MovieLens dataset, we have movie ratings, meaning that the users gave explicit feedback. So, we had to decide how to adapt the explicit feedback to implicit. A decision was made to keep ratings that are more than or equal to 4 as positive items, and ratings below 4 and unrated movies as negative items.

The loss function used in the BPR-Opt implementation is negative log-likelihood of the logistic sigmoid of the score difference between a positive item and negative item. Mathematically for a triplet (u, i, j) where u user prefers i item over j item, the loss is:

$$\text{Loss} = -\ln \sigma(\hat{x}_{ui} - \hat{x}_{uj}) + \lambda_\Theta ||\Theta||^2$$

Where:
$\hat{x}_{ui} - \hat{x}_{uj}$ - the score difference;
σ - the logistic sigmoid function $\frac{1}{1+e^{-x}}$;
ln - the natural logarithm;
$\lambda_\Theta ||\Theta||^2$ - the L_2 regularization term to prevent overfitting.

For optimization, Stochastic Gradient Descent with bootstrap sampling (randomly picking triplets to handle a large number of possible combinations) is applied.

As a result, we've got the BPR-Opt model with Precision@10 - 0.1625, that is very good score compared to the other models, Recall@10 - 0.0488, NDCG@10 - 0.2116.

Comparison of the BPR-Opt model to the Popularity-based heuristic model highlights the trade-off between the personalized model and the safe choice of popular items.

Popularity-based is a static calculation, while BPR-Opt requires iterative optimization. At early stages, BPR-Opt learns to recommend popular items, but at later stages, the model is trained to catch personalizations. But if the model is trained too long and without regularization, it can be prone to overfitting.

Popularity heuristic is sensitive to data volumes, but not sensitive to sampling techniques, while BPR-Opt is highly sensitive to negative sampling strategy. F.e., to learn certain user's taste, BPR-Opt have to receive samples with movies, that are popular, but this certain user did not watch.

The most significant difference between the two models is impact on head vs tail items. Popularity-based model learns to show head items, but tail items are mathematically suppressed. BPR-Opt model balances head items and promotes tail items by learning that a specific user prefers nich movie over popular one.

5 Online Evaluation: A/B Testing & Bandits

Except for calculating rating prediction and ranking metrics offline, to decide whether a recommendation system brings benefits, we have to conduct an online evaluation. Offline metrics are insufficient because it can only validate recommendations against movies the user already interacted with, ignoring potential popular movies the user never saw. It suffers from selection bias, as the historical data is skewed by the specifics of the previous logic used to collect it. Online evaluation is necessary because it captures real-time user interactions, allowing the model to receive immediate feedback on new or diverse items. Ultimately, only online testing can measure true business impact, such as long-term user retention and satisfaction. Online evaluation can be handled via A/B Testing or Multi-Armed Bandits.

5.1 A/B Testing

As a unit of randomization, we might consider session-level, user-level, and item-level. The main point is that we don't have session IDs in the dataset. But also, session-level randomization might add cross-exposure bias, which occurs when the same user is exposed to multiple variants during the experiment. For item-level randomization, it would be harder to attribute performance to a certain model (however, such kind of randomization can be applied as interleaving test). Anyway, for simplification and higher precision, for that test, we decided to stop on user-level randomization.

As a comparison metric, we chose Click-Through Rate (CTR), which will be simulated in our case by the availability of movie ratings in the test dataset. For the quardrail metrics we decided to calculate Average Rating of Recommendations (ARR). ARR will balance the recommendation comparison, ensuring that the quality of recommendations is not suffering while we are optimizing CTR.

For the evaluation simulation we chose two models, which on the offline metric Precision@10 showed very similar results. The models are from the similarity-based collaborative filtering algorithms: Cosine similarity user-based and Pearson similarity user-based. For the significance check, we applied T-test methodology.

A/B testing showed the next results: Cosine similarity user-based model CTR 9.87%, Pearson similarity user-based model CTR 1.39%. Metric difference is 8.48pp, p-value is 0.0000, so the difference is statistically significant.

Quardrail metric ARR showed inversely proportional results: Cosine similarity user-based model ARR is 4.3392 and Pearson similarity user-based model ARR is 4.4706. So, the metric difference is -0.1313, but p-value is 0.5875, so the difference is not statistically significant.

But there are several risks in A/B testing application for online evaluation in a real environment. Novelty effects can skew results because users may interact with an item simply because it is new, rather than because it is actually better. Position bias often leads to false positives, as items placed higher or more prominently on a screen naturally receive more clicks regardless of their true relevance. Feedback loops create a popular gets more popular scenario where models trained on biased A/B data continue to recommend the same items, narrowing the system's focus over time. These combined factors mean that a winning variant in a short-term test might actually worsen the user experience or catalog diversity in the long run. To deal with those risks, Multi-Armed Bandits can be applied for the models' online evaluation.

5.2 Multi-Armed Bandits

We decided to compare the performance of the same CF models (Cosine similarity user-based and Pearson similarity user-based) applying Multi-Armed bandits strategies: ϵ-greedy and Thompson Sampling. Also, we estimated static policy results to evaluate against the strategies. As a result, the mean CTR in ϵ-greedy bandits simulation is 9.27%, models' distribution is cosine similarity user-based 287 times and pearson similarity user-based 13 times. In Thompson Sampling bandits simulation CTR is 9.14%, models distribution is 294 and 6 times respectively. For static policy simulation, we used Cosine similarity user-based model, which resulted in 9.17% CTR. In our case ϵ-greedy strategy became a winner.

The core of Multi-Armed Bandit strategies is exploration vs exploitation trade-off. Exploitation is picking the current best-performing model to maximize immediate clicks, while Exploration is trying other models to see if they might actually be better. The ϵ-greedy strategy balances this by mostly picking the winner but reserving a fixed percentage of time to choose a model completely at random. In contrast, Thompson Sampling uses a probabilistic approach: it models each model's performance as a distribution and picks based on which one is likely to be best. While ϵ-greedy explores blindly at a constant rate, Thompson Sampling naturally explores less as it becomes more confident in which model is superior, making it more efficient over time.

6 Final System-Level Synthesis

As a production engineer, our priority would be not a slight boost in ranking metrics, but the recommendation system reliability, data integrity, and the feedback loop.

6.1 Offline vs Online Discrepancies

The offline vs online discrepancies are the most probable reason production systems can fail. Offline, it's easy to accidentally include future data in user histories. In production, features must be strictly point-in-time. Besides MovieLens data has plenty of ratings for every movie. In production, new movies might arrive daily with zero interactions. Offline metrics won't show us that the models are likely ignoring new movies. Another possible problem is latency: similarity-based CF models might have high offline accuracy, but the inference time for such models is too long.

6.2 Deployment Choice and Justification

The chosen deployment strategy would be a phased rollout consisting of shadow deployment followed by a user-level randomized A/B test. In the shadow phase, the new recommender model would run in parallel with the production model to validate data pipelines, prediction stability, and latency without affecting users. After technical validation, a controlled A/B test is conducted to measure the impact of the new model on online engagement while monitoring guardrail metrics such as latency or churn. This approach is justified because offline metrics alone cannot guarantee real-world performance; only randomized online experimentation provides reliable evidence of user impact while minimizing operational and business risks. In later stages, adaptive traffic allocation methods using multi-armed bandits strategies would be considered to optimize cumulative performance.

6.3 Iteration Strategy Post-Deployment

Post-deployment, the iteration strategy relies on continuous monitoring, controlled experimentation, and periodic model updates. After rollout, key performance metrics and guardrails have to be tracked both at aggregate and segment levels to detect degradation, drift, or unintended side effects such as reduced diversity. If performance declines or new improvement opportunities are identified, incremental updates—such as retraining, hyperparameter tuning, or feature enhancements have to be introduced and validated through offline and online evaluation. This iterative approach ensures that the recommender system adapts to evolving user behavior while maintaining stability, reliability, and measurable business impact.

6.4 Key Failure Modes to Monitor

Key failure modes to monitor include feedback loops, position bias, distribution drift, and system instability. Feedback loops may strengthen popularity bias by repeatedly promoting already popular items and reducing diversity. Position bias is a tendency of users to interact more with items shown at higher ranks in a recommendation list, regardless of their true relevance. It can distort interaction signals, leading to misleading performance estimates. Distribution drift may gradually decrease model quality as user preferences and assortment change over time. Additionally, operational risks such as increased latency or error rates must be tracked, as improvements in predictive accuracy are not meaningful if system reliability is compromised. Continuous monitoring of these risks is essential to ensure sustainable and trustworthy recommender system performance.