DATA SCIENCE & ML

Recommender Systems at Scale

Collaborative filtering, embeddings and multi-stage ranking

Houssam Kodad

PDF · DATAFORGE BOOKS

Questions about this sample or the full edition: **support@dataforgebooks.com**

CONTENTS

Table of Contents

CHAPTER 01

The Anatomy of a Recommender

Recommendations quietly drive an enormous share of engagement and revenue across modern products, yet the distance between a textbook matrix factorisation and a production recommender is vast. The textbook version scores every item for every user; the production version cannot, because there are millions of items and the answer is needed in tens of milliseconds. Bridging that gap is what this book is about.

This chapter lays out the real architecture. We walk through the retrieve-rank-rerank pattern that makes recommendation tractable at scale, the crucial and often-misunderstood difference between implicit and explicit feedback, and the business objectives that hide behind a simple click. By the end you will see a recommender not as a single model but as a pipeline of specialised stages, each with its own trade-offs.

We also confront, from the start, the uncomfortable truths that make recommenders hard: feedback is biased by what you already showed, the metric you train on is only a proxy for what you want, and a system that optimises clicks alone will cheerfully make your product worse. Facing these early is what separates a recommender that helps users from one that games a number.

The Anatomy of a Recommender

Production recommenders are almost always multi-stage, and the reason is brutally practical: you cannot run your best model over an entire catalogue in real time. So the work is split. A fast retrieval stage narrows millions of candidates to a few hundred using cheap approximate methods. A ranking stage then scores those few hundred with a richer, more expensive model. Finally a re-ranking stage applies business rules — diversity, freshness, de-duplication — to the handful that reach the user.

This staging is the organising idea of the entire book. By spending cheap compute to assemble a good shortlist and expensive compute only on that shortlist, you capture most of the quality at a fraction of the cost and latency. Each stage has different goals, different models, and different failure modes, and most of what follows is a detailed tour of how to build each one well.

Implicit vs Explicit Feedback

Explicit feedback — star ratings, thumbs up and down — is precious but rare, because most users never rate anything. Implicit feedback — clicks, watches, purchases, dwell time — is abundant but noisy. A click is not an endorsement, and, crucially, the absence of a click is not rejection: the user

may simply never have seen the item. Real systems run almost entirely on implicit signals, and handling their ambiguity well is a recurring theme.

The subtlety that trips up newcomers is that "no interaction" usually means "no exposure," not "no interest." Treating every non-click as a negative example teaches the model that users dislike things they were never shown. Proper handling — confidence weighting, careful negative sampling, modelling exposure — is what makes implicit-feedback recommenders work, and we develop it carefully rather than waving it away.

Collaborative Filtering Foundations

The oldest and still one of the most powerful ideas in recommendation is collaborative filtering: users who agreed in the past will agree in the future. Its classic form, matrix factorisation, learns a compact vector for each user and each item such that their dot product approximates the interaction, discovering latent tastes nobody labelled — that this user likes a certain genre, that this item appeals to a certain segment.

Matrix factorisation is worth understanding deeply because it is both a strong baseline and the conceptual seed of the embedding methods that dominate modern systems. Its limitations — cold start, sparsity, the difficulty of using side information — are exactly the problems later techniques were invented to solve, so it is the right place to begin and the reference everything else is measured against.

Learning Embeddings

Modern retrieval replaces hand-tuned similarity with learned embeddings: dense vectors for users and items, trained so that relevant pairs sit close together in the vector space. The dominant architecture is the two-tower model — one network encodes the user, another encodes the item, and the system is trained so a user's vector is near the items they engaged with and far from those they did not.

Two-tower models are popular because they fit retrieval perfectly: item vectors can be computed in advance and indexed, so at request time you embed the user once and find nearby items in milliseconds. The art is in the training, especially negative sampling — choosing which non-interacted items to push away from the user. Get that right and retrieval is excellent; get it wrong and the model learns to separate popular from unpopular rather than relevant from irrelevant.

```
# Two-tower retrieval: precompute item vectors, embed the user at request
# time, and find nearest items by dot product.
user_vec  = user_tower(user_features)            # one embedding per request
scores    = item_vectors @ user_vec              # millions of dot products
top_k     = ann_index.search(user_vec, k=200)    # approximate, sub-ms
```

Candidate Generation at Scale

Retrieval has to find the best few hundred items out of millions, fast, and it does so with approximate nearest-neighbour search. Exact search over millions of vectors per request is too slow, so ANN indexes trade a tiny amount of recall for orders of magnitude in speed, finding almost all of the truly nearest items in well under a millisecond.

Real systems rarely rely on a single retrieval source. They blend several — a two-tower model, a popularity source, a recently-trending source, a graph of co-purchases — because each captures a different notion of relevance and together they cover for one another's blind spots. Building, refreshing and combining these candidate sources is a substantial engineering effort that we treat as the serious systems problem it is.

Ranking and Re-Ranking

Once retrieval has produced a few hundred candidates, the ranking model scores them with far richer features than retrieval can afford — detailed user history, item attributes, context, and cross-features between them. Because it runs on hundreds of items rather than millions, it can be a heavier, more accurate model, and it is typically where the bulk of measured quality improvement comes from.

Re-ranking is the final, business-aware pass over the ranked list. Pure relevance is not enough: users want variety, freshness matters, the same item should not appear twice, and hard constraints like inventory or policy must be honoured. Re-ranking enforces diversity and rules on the top results, and it is often where a technically good recommender is turned into one that actually feels good to use.

The Metric Is Only a Proxy

Optimising for clicks alone is a trap that every recommendation team must consciously avoid. Clickbait gets clicks; it does not get satisfied users or long-term revenue. The metric you train on is always a proxy for the outcome you actually want, and a sufficiently capable recommender will ruthlessly exploit any gap between the two, degrading the product while its offline numbers improve.

Mature systems therefore balance multiple objectives — engagement, satisfaction, diversity, long-term retention, revenue — and, decisively, they measure success with online experiments rather than offline proxies alone. Keeping the true objective in view, and remembering that clicks are merely a stand-in for it, is the judgement that separates a recommender that serves the business from one that quietly games a dashboard. We return to it throughout.

This is a free sample

You've reached the end of the sample chapter.

Get the complete book — every chapter, fully worked — at **dataforgebooks.com**.

FULL EDITION · 268 PAGES · PDF

Read the full title at **dataforgebooks.com**

Questions? support@dataforgebooks.com