Recommender Systems for Tailored Testing (R Tutorial)

Kelly D. Edwards, Ph.D.

Contents

Purpose. This tutorial shows how to use collaborative filtering with **matrix factorization (Funk SVD)** to predict item *relevance ratings* in a longitudinal assessment context.
Data. We use **simulated** participant × item ratings (1-4 scale). No real or operational data are used.

Introduction

Many medical boards are moving away from a single, high-stakes recertification exam and toward longitudinal assessments. Instead of one big test every 10 years, these programs deliver shorter sets of questions more frequently, often online. The goal is to reduce burden and better integrate assessment with day-to-day learning.

One ongoing challenge, however, is relevance. Not every question feels meaningful to every physician. For example, a community physician might encounter a question about a rare inpatient procedure that doesn't reflect their daily practice, even though the topic is clinically important overall.

Research in the learning sciences shows that when questions feel irrelevant:

- Engagement tends to drop
- Motivation to learn decreases
- The value of certification itself can be questioned

A Potential Solution: Recommender Systems

One way to address this challenge is through recommender systems, which are the same type of algorithms that suggest movies on Netflix or products on Amazon.

Here's how they can be applied in an assessment setting:

- Physicians provide feedback (e.g., rating how relevant each question felt to their practice)
- A collaborative filtering algorithm looks for patterns across thousands of ratings
- The system predicts which future questions are most likely to feel relevant for each physician
- Items are then delivered based on these predictions, while still respecting content balance and fairness requirements

This approach has the potential to make longitudinal assessments feel more personalized and engaging, while still preserving the rigor needed for certification.

R Tutorial

To give others a chance to see how this works, I've created a tutorial with example R code using the `recommenderlab` package. The tutorial walks through:

- Simulating item relevance ratings,
- Training a recommender system with Funk SVD, and
- Generating predictions for new participants.

Setup

```
set.seed(2025)

# Core packages
library("recommenderlab")
library("Matrix")
library("ggplot2")
```

If `recommenderlab` is not installed:

```
install.packages("recommenderlab")
```

Why Funk SVD? In sparse ratings matrices (many missing values), matrix factorization learns low-rank latent factors for participants and items, allowing us to estimate unobserved ratings (i.e., how relevant a new item is likely to feel).

Simulate a participant × item relevance matrix

We'll simulate:

- 600 participants (rows)
- 200 items (cols)

- 1-4 relevance ratings, with ~70% missingness (no participants see all items)

We also simulate latent "practice profiles" and item themes so the data have real structure (e.g., participants with a "primary care" profile tend to rate certain items as more relevant).

```
n_users  <- 600
n_items  <- 200
missing_rate <- 0.7   # ~70% missing
K <- 8                # latent dimensions

# Simulate latent factors for users/items
U <- matrix(rnorm(n_users * K, sd = 0.7), n_users, K)     # users
V <- matrix(rnorm(n_items * K, sd = 0.7), n_items, K)     # items

# Affinity = U %*% t(V)
affinity <- U %*% t(V)

# Map continuous affinity to 1..4 ratings with noise
eps <- matrix(rnorm(n_users * n_items, sd = 0.6), n_users, n_items)
score <- scale(affinity + eps)            # standardize

# breakpoints for 1..4
q <- quantile(score, probs = c(.25, .5, .75))
to_rating <- function(x, q) {
  cut(x, breaks = c(-Inf, q[1], q[2], q[3], Inf), labels = 1:4)
}
ratings_full <- matrix(as.integer(to_rating(score, q)), n_users, n_items)

# Add missingness at random
mask <- matrix(runif(n_users * n_items) < missing_rate, n_users, n_items)
ratings <- ratings_full
ratings[mask] <- NA_integer_

dim(ratings); mean(is.na(ratings))
```

```
## [1] 600 200
```

```
## [1] 0.7005583
```

Convert to a `recommenderlab` object:

```
R <- as(ratings, "realRatingMatrix")
R
```

```
## 600 x 200 rating matrix of class 'realRatingMatrix' with 35933 ratings.
```

Train/test split & evaluation scheme

We'll hold out a test set of 150 users. Within the training set, we'll use an `evaluationScheme` to compute RMSE/MAE.

```r
idx_test  <- sample(1:n_users, size = 150)
R_test    <- R[idx_test, ]
R_train   <- R[-idx_test, ]

# evaluationScheme: 80/20 split on known ratings, 5-fold CV
es <- evaluationScheme(R_train, method = "cross-validation", k = 5, given = -1, goodRating = 4)
es
```

```
## Evaluation scheme using all-but-1 items
## Method: 'cross-validation' with 5 run(s).
## Good ratings: >=4.000000
## Data set: 450 x 200 rating matrix of class 'realRatingMatrix' with 27005 ratings.
```

Note: given = -1 uses all available ratings for training folds; we evaluate on the held-out portion within each fold.

Fit Funk SVD with `recommenderlab`

`recommenderlab` exposes SVD-based recommenders via `method = "SVD"` (Funk-style gradient descent). We'll compare a simple baseline (POPULAR) against SVD.

```r
algos <- list(
  "POPULAR" = list(name = "POPULAR", param = NULL),
  "SVD_k20" = list(name = "SVD",     param = list(k = 20, maxiter = 200, normalize = "center")),
  "SVD_k40" = list(name = "SVD",     param = list(k = 40, maxiter = 200, normalize = "center"))
)

results <- evaluate(es, method = algos, type = "ratings")
```

```
## POPULAR run fold/sample [model time/prediction time]
##   1  [0.022sec/0.079sec]
##   2  [0.006sec/0.006sec]
##   3  [0.005sec/0.006sec]
##   4  [0.004sec/0.018sec]
##   5  [0.004sec/0.007sec]
## SVD run fold/sample [model time/prediction time]
##   1  [0.051sec/0.011sec]
##   2  [0.059sec/0.01sec]
##   3  [0.046sec/0.008sec]
##   4  [0.05sec/0.017sec]
##   5  [0.051sec/0.009sec]
## SVD run fold/sample [model time/prediction time]
##   1  [0.088sec/0.016sec]
##   2  [0.091sec/0.008sec]
##   3  [0.115sec/0.006sec]
##   4  [0.083sec/0.008sec]
##   5  [0.131sec/0.008sec]
```

```r
# Summarize RMSE/MAE across folds
perf <- lapply(results, function(res) {
  data.frame(
```

```
    RMSE = avg(res, "RMSE"),
    MAE  = avg(res, "MAE")
  )
})
do.call(rbind, perf)
```

```
##         RMSE.RMSE RMSE.MSE  RMSE.MAE  MAE.RMSE  MAE.MSE   MAE.MAE
## POPULAR 1.1399459 1.301450 1.0149085 1.1399459 1.301450 1.0149085
## SVD_k20 0.9780573 0.957810 0.8612992 0.9780573 0.957810 0.8612992
## SVD_k40 1.0067870 1.014651 0.8755282 1.0067870 1.014651 0.8755282
```

Tip: Increase `k` for more latent dimensions at the cost of potential overfit; use `CV` to pick `k`.

Train final model and generate predictions

Train on all training users, then predict ratings for the held-out test users.

```
rec <- Recommender(R_train, method = "SVD", parameter = list(k = 30, maxiter = 200, normalize = "center
pred_test <- predict(object = rec, newdata = R_test, type = "ratingMatrix")
pred_mat  <- as(pred_test, "matrix")   # numeric predictions
true_mat  <- as(R_test, "matrix")      # ground truth with NAs

# Evaluate on co-observed cells
co_obs <- !is.na(true_mat) & !is.na(pred_mat)
rmse <- sqrt(mean((pred_mat[co_obs] - true_mat[co_obs])^2, na.rm=T))
mae  <- mean(abs(pred_mat[co_obs] - true_mat[co_obs]), na.rm=T)
c(RMSE = rmse, MAE = mae)
```

```
##      RMSE       MAE
## 0.7973839 0.6867974
```

Inspect a few predictions:

```
pred_mat[1:6, 1:8]
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]     [,8]
## [1,] 2.442729 2.831136 2.113613 3.040974 2.709425 1.605645 2.753223 2.025875
## [2,] 2.946876 2.878281 2.821849 2.641069 2.581239 2.956996 2.609371 2.742959
## [3,] 2.467677 1.925212 2.501307 2.319656 2.469397 2.307568 2.547723 2.208881
## [4,] 2.618050 2.906663 2.427284 2.749667 2.851427 2.787904 2.051954 2.880361
## [5,] 2.306464 2.275101 2.698023 2.209545 2.278022 2.499635 2.092093 2.360601
## [6,] 2.754446 1.601045 2.192409 2.039755 2.548205 2.208795 2.570957 2.051773
```

From predictions to tailored selection

In an assessment setting, we don't simply "send the top-N" items by prediction. We post-filter to respect constraints:

- Blueprint coverage: maintain required domain proportions

- Exposure control: cap how often items are seen
- Item quality: only deliver items meeting psychometric standards
- Candidate eligibility: remove items already seen by the participant

Here's a toy post-filtering function that enforces a (simplified) domain mix.

```
# Simulate item domains
domains <- factor(sample(c("Cardio","Neuro","ID","Endo","Pulm"), n_items, replace = TRUE))

select_tailored <- function(pred_row, seen_idx = integer(0), n_select = 20,
                            domains, target_mix = c(Cardio=0.2, Neuro=0.2, ID=0.2, Endo=0.2, Pulm=0.2))
  # Remove already-seen items
  ok <- setdiff(which(!is.na(pred_row)), seen_idx)
  cand <- data.frame(item = ok, pred = pred_row[ok], domain = domains[ok], stringsAsFactors = FALSE)
  cand <- cand[order(-cand$pred), ]

  # Greedy fill by domain proportions
  target_counts <- round(target_mix * n_select)
  out <- integer(0)
  for (d in names(target_counts)) {
    need <- target_counts[d]
    pool <- cand[cand$domain == d & !(cand$item %in% out), ]
    take <- head(pool$item, need)
    out <- c(out, take)
  }
  # If not enough in a domain, top-up from remaining highest predictions
  if (length(out) < n_select) {
    extra <- setdiff(cand$item, out)
    out <- c(out, head(extra, n_select - length(out)))
  }
  out
}

# Example: pick 20 items for test user 1
sel_items <- select_tailored(pred_mat[1, ], seen_idx = which(!is.na(true_mat[1, ])), n_select = 20, doma
length(sel_items); head(sel_items)
```

```
## [1] 20
```

```
## [1]  89  65  79   2 172 182
```

Interpreting predictions (face validity checks)

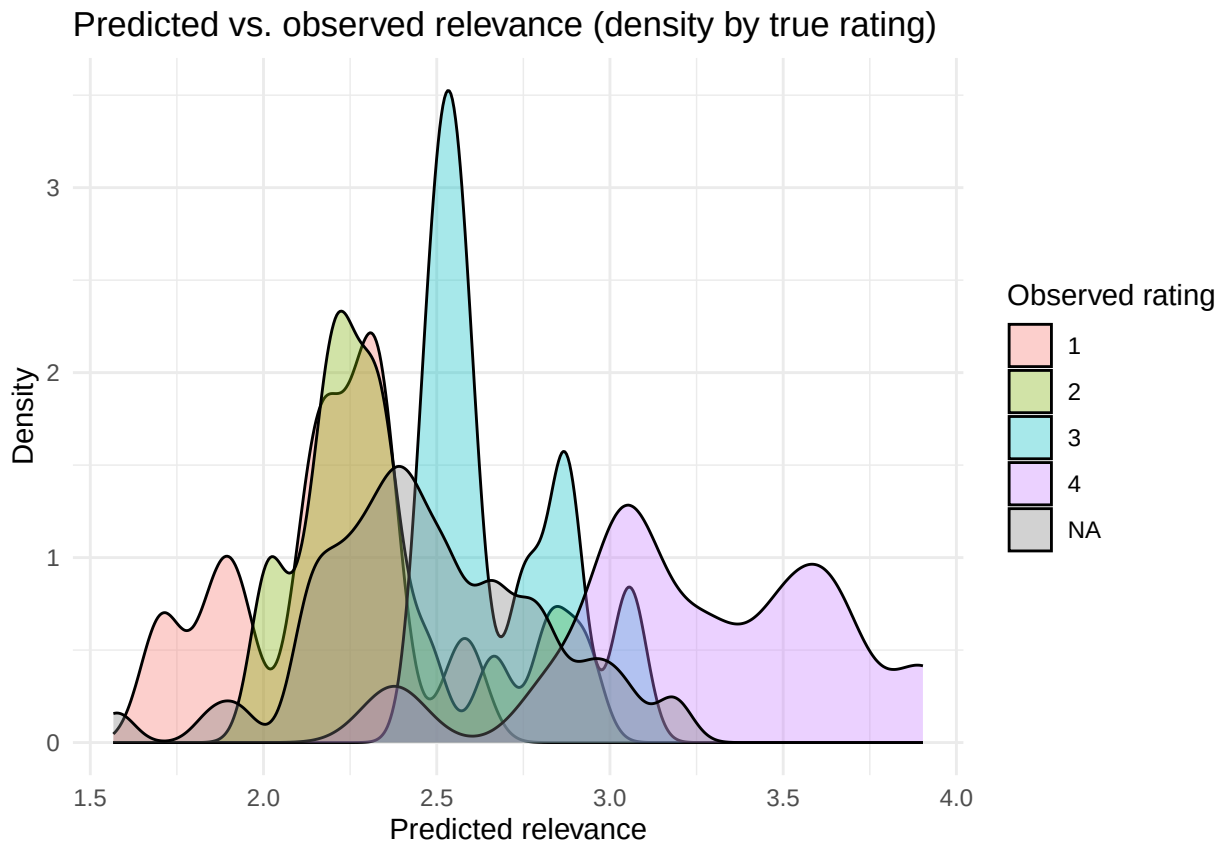
It helps to visualize how predictions separate 1–4 "true" ratings.

```
# Sample co-observed cells for plotting
co_idx <- which(co_obs, arr.ind = TRUE)
samp   <- co_idx[sample(nrow(co_idx), size = min(8000, nrow(co_idx))), drop = FALSE]

df_plot <- data.frame(
  pred = pred_mat[samp],
```

```
  true = factor(true_mat[samp], levels = 1:4)
)

ggplot(df_plot, aes(x = pred, fill = true)) +
  geom_density(alpha = 0.35) +
  labs(x = "Predicted relevance", y = "Density", fill = "Observed rating",
       title = "Predicted vs. observed relevance (density by true rating)") +
  theme_minimal()
```



You should see higher predictions shifting toward higher observed ratings—i.e., monotonic separation.

Cold-start and drift (practical considerations)

- Item cold-start: New items with no ratings can borrow information from content features (e.g., tags, blueprint domain) via hybrid models, or by seeding with pilot ratings.
- User cold-start: For brand-new participants, use population priors, brief "warm-up" questions, or side info (practice profile) to initialize.
- Concept drift: Re-train on a cadence (e.g., quarterly) and monitor prediction error over time.

Fairness & psychometric guardrails (assessment context)

Even with good accuracy, governance matters. In assessments:

- Blueprint adherence: Item selection must guarantee domain coverage.
- Score comparability: If scores are used, maintain comparability via equating/linking; personalization must not change score meaning.
- Fairness checks: Routinely evaluate subgroup RMSE/MAE and calibration (e.g., by practice setting, geography, training cohort). Investigate/mitigate gaps.
- Exposure & security: Cap exposures, control for leakage, and rotate forms/items appropriately.
- Transparency: Document the algorithm, inputs, re-training cadence, and monitoring plan.

What "good enough" looks like

No model is perfect. You want:

- Consistent lift over baseline (e.g., SVD vs. POPULAR) on RMSE/MAE.
- Face-valid separation of predictions by observed rating.
- Operational impact: higher average relevance after deployment without violating blueprint/fairness constraints.