Project #4: Evaluating and Enhancing Recommender Systems

Saloua Daouki

2025-06-24

Contents

Introduction

Recommender systems are essential in modern digital platforms, guiding users toward relevant content and increasing engagement. This project investigates the evaluation and comparison of different recommender

algorithms, implements a business-relevant goal (diversity), and critically reflects on the challenges of offline evaluation—drawing on *Building a Recommendation System in R* (Chapters 4 and 5) and real-world practices.

Literature Insights

Evaluation Methodology (Chapter 4):

Effective recommender system evaluation involves splitting data into training and test sets, hiding user preferences, and predicting them to simulate real-world use. The choice of metrics and methods should align with business goals—accuracy alone may not suffice if the aim is engagement or diversity.

Case Study Approach (Chapter 5):

A real-world case study demonstrates transforming raw interaction logs into a user-item matrix, building and optimizing models, and evaluating them. Notably, the MovieLens 100k dataset used here is already provided as a user-item rating matrix, allowing immediate application of collaborative filtering models in `recommenderlab` without data conversion.

Data Preparation

```
# Load MovieLens 100k dataset
data(MovieLense)
ratings <- MovieLense
dim(ratings)
```

```
## [1]  943 1664
```

```
summary(rowCounts(ratings))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    19.0    32.0    64.0   105.4   147.5   735.0
```

```
summary(colCounts(ratings))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    7.00   27.00   59.73   80.00  583.00
```

1. Comparing Two Recommender Algorithms

I split the data, simulate hiding preferences, and compare the accuracy of user-based (UBCF) and item-based (IBCF) collaborative filtering.

```
set.seed(123)
eval_scheme <- evaluationScheme(ratings, method="split", train=0.8, given=10, goodRating=4)
```

A. User-Based Collaborative Filtering (UBCF)

```
ubcf_model <- Recommender(getData(eval_scheme, "train"), method="UBCF")
ubcf_pred <- predict(ubcf_model, getData(eval_scheme, "known"), type="ratings")
ubcf_acc <- calcPredictionAccuracy(ubcf_pred, getData(eval_scheme, "unknown"))
ubcf_acc
```

```
##      RMSE       MSE       MAE
## 1.2108100 1.4660609 0.9529885
```

B. Item-Based Collaborative Filtering (IBCF)

```
ibcf_model <- Recommender(getData(eval_scheme, "train"), method="IBCF")
ibcf_pred <- predict(ibcf_model, getData(eval_scheme, "known"), type="ratings")
ibcf_acc <- calcPredictionAccuracy(ibcf_pred, getData(eval_scheme, "unknown"))
ibcf_acc
```

```
##      RMSE       MSE       MAE
## 1.0576100 1.1185389 0.7572206
```

Accuracy Comparison

Algorithm	RMSE	MAE
UBCF	1.2108	0.953
IBCF	1.0576	0.7572

2. Beyond Accuracy: Business/User Goal—Diversity

Diversity in recommendations can increase user satisfaction and engagement. I measure intra-list diversity (how different the recommended items are from each other).

```
# Diversity calculation
calc_diversity <- function(topNList, sim_matrix) {
  lists <- as(topNList, "list")
  diversities <- sapply(lists, function(items) {
    if(length(items) <= 1) return(NA)
    sim_values <- sim_matrix[items, items]
    sim_values[lower.tri(sim_values, diag = TRUE)] <- NA
    mean(1 - sim_values, na.rm = TRUE)
  })
  mean(diversities, na.rm = TRUE)
}

top_n <- 5
ubcf_topN <- predict(ubcf_model, getData(eval_scheme, "known"), type="topNList", n=top_n)
item_sim <- similarity(ratings, method = "jaccard", which = "items")
div_ubcf <- calc_diversity(ubcf_topN, as.matrix(item_sim))
div_ubcf
```

```
## [1] 0
```

Diversity-Enhanced (Hybrid) Recommender

I combine UBCF with a random recommender to boost diversity.

```
random_model <- Recommender(getData(eval_scheme, "train"), method="RANDOM")
hybrid_model <- HybridRecommender(ubcf_model, random_model, weights = c(0.7, 0.3))
hybrid_topN <- predict(hybrid_model, getData(eval_scheme, "known"), type="topNList", n=top_n)
div_hybrid <- calc_diversity(hybrid_topN, as.matrix(item_sim))
div_hybrid
```

```
## [1] 0
```

3. Accuracy vs. Diversity Comparison

I compare hit rate (accuracy) and diversity for UBCF and the hybrid model.

```
hit_rate <- function(pred, actual, n) {
  pred_items <- as(pred, "list")
  actual_items <- as(actual, "list")
  hits <- mapply(function(p, a) length(intersect(p, a)), pred_items, actual_items)
  mean(hits / n)
}
actual_unknown <- getData(eval_scheme, "unknown")
hit_ubcf <- hit_rate(ubcf_topN, actual_unknown, top_n)
hit_hybrid <- hit_rate(hybrid_topN, actual_unknown, top_n)
accuracy_df <- data.frame(
  Method = c("UBCF", "Hybrid (Diverse)"),
  HitRate = c(hit_ubcf, hit_hybrid),
  Diversity = c(div_ubcf, div_hybrid)
)
knitr::kable(accuracy_df, digits = 4, caption = "Hit Rate and Diversity Comparison")
```

Table 2: Hit Rate and Diversity Comparison

Method	HitRate	Diversity
UBCF	0	0
Hybrid (Diverse)	0	0

4. The Limits of Offline Evaluation

Offline metrics (accuracy, diversity, etc.) help compare models before deployment, but do not capture user engagement or satisfaction.

Online-Only Experiments:

- A/B test different recommenders on real users
- Measure click-through rate, session time, and explicit feedback
- Randomize user assignment and ensure robust tracking

5. Expanded Metrics: Precision, Recall, Novelty, Diversity, Serendipity

I expand the evaluation with additional metrics for a more holistic view.

5.1. Evaluation Setup

Note: As seen below, all models scored zero for precision/recall under these splits, which is explored in detail in the diagnostics and lessons learned sections.

```
# Only train the models once and reuse them throughout
set.seed(123)
scheme <- evaluationScheme(MovieLense, method = "split", train = 0.8, given = 10, goodRating = 4)
top_n <- 5
ubcf_model <- Recommender(getData(scheme, "train"), method = "UBCF")
```

```
svd_model  <- Recommender(getData(scheme, "train"), method = "SVD", parameter = list(k = 20))
reg_model  <- Recommender(getData(scheme, "train"), method = "POPULAR")
```

5.2. Top-N Predictions

```
ubcf_topN <- predict(ubcf_model, getData(scheme, "known"), type = "topNList", n = top_n)
svd_topN  <- predict(svd_model,  getData(scheme, "known"), type = "topNList", n = top_n)
reg_topN  <- predict(reg_model,  getData(scheme, "known"), type = "topNList", n = top_n)
actual_unknown <- getData(scheme, "unknown")
```

5.3. Precision and Recall

Precision:

Measures the proportion of recommended items that are actually relevant to the user. Higher precision means the recommendations are more accurate.

Recall:

Measures the proportion of relevant items that were successfully recommended out of all possible relevant items for the user. Higher recall means more of what the user likes is found by the model.

```
precision_recall <- function(pred, actual, n) {
  pred_list <- as(pred, "list")
  actual_list <- as(actual, "list")
  precisions <- mapply(function(p, a) if(length(p)) length(intersect(p, a))/n else NA, pred_list, actual
  recalls    <- mapply(function(p, a) if(length(a)) length(intersect(p, a))/length(a) else NA, pred_list
  c(Precision = mean(precisions, na.rm = TRUE), Recall = mean(recalls, na.rm = TRUE))
}
pr_ubcf <- precision_recall(ubcf_topN, actual_unknown, top_n)
pr_svd  <- precision_recall(svd_topN,  actual_unknown, top_n)
pr_reg  <- precision_recall(reg_topN,  actual_unknown, top_n)
```

5.4. Novelty

Indicates how uncommon or unexpected the recommended items are. Higher novelty means the system suggests less popular, more unique items, helping users discover new content.

```
item_counts <- colCounts(MovieLense)
item_popularity <- item_counts / max(item_counts)
calc_novelty <- function(topNList, item_pop) {
  rec_list <- as(topNList, "list")
  mean_novelty <- sapply(rec_list, function(items) {
    if(length(items)==0) return(NA)
    1 - mean(item_pop[items], na.rm = TRUE)
  })
  mean(mean_novelty, na.rm = TRUE)
}
nov_ubcf <- calc_novelty(ubcf_topN, item_popularity)
nov_svd  <- calc_novelty(svd_topN,  item_popularity)
nov_reg  <- calc_novelty(reg_topN,  item_popularity)
```

5.5. Diversity

Measures how different the recommended items are from each other. Higher diversity means the recommendation list contains a wider variety of items, reducing redundancy.

```
item_sim <- similarity(MovieLense, method = "jaccard", which = "items")
calc_diversity <- function(topNList, sim_matrix) {
  lists <- as(topNList, "list")
  diversities <- sapply(lists, function(items) {
    if(length(items) <= 1) return(NA)
    sim_values <- sim_matrix[items, items]
    sim_values[lower.tri(sim_values, diag = TRUE)] <- NA
    mean(1 - sim_values, na.rm = TRUE)
  })
  mean(diversities, na.rm = TRUE)
}
div_ubcf <- calc_diversity(ubcf_topN, as.matrix(item_sim))
div_svd  <- calc_diversity(svd_topN,  as.matrix(item_sim))
div_reg  <- calc_diversity(reg_topN,  as.matrix(item_sim))
```

5.6. Serendipity

Captures how many recommended items are both relevant and surprising (not just popular or obvious choices). High serendipity means the system can delight users with unexpected but welcome suggestions.

```
pop_threshold <- quantile(item_counts, 0.9)
unpopular_items <- names(item_counts[item_counts < pop_threshold])
calc_serendipity <- function(topNList, actual, unpopular_items) {
  rec_list <- as(topNList, "list")
  actual_list <- as(actual, "list")
  sers <- mapply(function(rec, act) {
    sum(rec %in% act & rec %in% unpopular_items) / length(rec)
  }, rec_list, actual_list)
  mean(sers, na.rm = TRUE)
}
ser_ubcf <- calc_serendipity(ubcf_topN, actual_unknown, unpopular_items)
ser_svd  <- calc_serendipity(svd_topN,  actual_unknown, unpopular_items)
ser_reg  <- calc_serendipity(reg_topN,  actual_unknown, unpopular_items)
```

5.7. Metrics Summary Table

```
metrics_df <- data.frame(
  Model = c("User-Based CF", "SVD", "Popularity"),
  Precision = c(pr_ubcf["Precision"], pr_svd["Precision"], pr_reg["Precision"]),
  Recall    = c(pr_ubcf["Recall"],    pr_svd["Recall"],    pr_reg["Recall"]),
  Novelty   = c(nov_ubcf, nov_svd, nov_reg),
  Diversity = c(div_ubcf, div_svd, div_reg),
  Serendipity = c(ser_ubcf, ser_svd, ser_reg)
)
knitr::kable(metrics_df, digits = 4, caption = "Offline Top-N Metrics: Precision, Recall, Novelty, Dive
```

Table 3: Offline Top-N Metrics: Precision, Recall, Novelty, Diversity, Serendipity

Model	Precision	Recall	Novelty	Diversity	Serendipity
User-Based CF	0	0	0.8700	0	0
SVD	0	0	0.9969	0	0
Popularity	0	0	0.2219	0	0

6. Revised Evaluation Scheme

Given the zero scores above, I revised the evaluation: more known ratings per user, removed the goodRating threshold, and applied cross-validation.

```
set.seed(123)
scheme2 <- evaluationScheme(
  MovieLense,
  method = "cross-validation",
  k = 5,
  given = 20,
  goodRating = NA
)
```

```
## Warning in .local(data, ...): Dropping these users from the evaluation since they have fewer rating
## These users are 19, 36, 140, 242, 300, 302, 309, 364, 475, 812, 824, 866, 873, 926
```

```
top_n2 <- 5
ubcf_model2 <- Recommender(getData(scheme2, "train"), method = "UBCF")
svd_model2  <- Recommender(getData(scheme2, "train"), method = "SVD", parameter = list(k = 20))
reg_model2  <- Recommender(getData(scheme2, "train"), method = "POPULAR")
ubcf_topN2 <- predict(ubcf_model2, getData(scheme2, "known"), type = "topNList", n = top_n2)
svd_topN2  <- predict(svd_model2,  getData(scheme2, "known"), type = "topNList", n = top_n2)
reg_topN2  <- predict(reg_model2,  getData(scheme2, "known"), type = "topNList", n = top_n2)
actual_unknown2 <- getData(scheme2, "unknown")
```

Metrics After Revision

```
precision_recall2 <- function(pred, actual, n) {
  pred_list <- as(pred, "list")
  actual_list <- as(actual, "list")
  precisions <- mapply(function(p, a) if(length(p)) length(intersect(p, a))/n else NA, pred_list, actual
  recalls    <- mapply(function(p, a) if(length(a)) length(intersect(p, a))/length(a) else NA, pred_list
  c(Precision = mean(precisions, na.rm = TRUE), Recall = mean(recalls, na.rm = TRUE))
}
pr_ubcf2 <- precision_recall2(ubcf_topN2, actual_unknown2, top_n2)
pr_svd2  <- precision_recall2(svd_topN2,  actual_unknown2, top_n2)
pr_reg2  <- precision_recall2(reg_topN2,  actual_unknown2, top_n2)

item_counts2 <- colCounts(MovieLense)
item_popularity2 <- item_counts2 / max(item_counts2)
calc_novelty2 <- function(topNList, item_pop) {
  rec_list <- as(topNList, "list")
  mean_novelty <- sapply(rec_list, function(items) {
    if(length(items)==0) return(NA)
```

```
    1 - mean(item_pop[items], na.rm = TRUE)
  })
  mean(mean_novelty, na.rm = TRUE)
}
nov_ubcf2 <- calc_novelty2(ubcf_topN2, item_popularity2)
nov_svd2  <- calc_novelty2(svd_topN2,  item_popularity2)
nov_reg2  <- calc_novelty2(reg_topN2,  item_popularity2)

item_sim2 <- similarity(MovieLense, method = "jaccard", which = "items")
calc_diversity2 <- function(topNList, sim_matrix) {
  lists <- as(topNList, "list")
  diversities <- sapply(lists, function(items) {
    if(length(items) <= 1) return(NA)
    sim_values <- sim_matrix[items, items]
    sim_values[lower.tri(sim_values, diag = TRUE)] <- NA
    mean(1 - sim_values, na.rm = TRUE)
  })
  mean(diversities, na.rm = TRUE)
}
div_ubcf2 <- calc_diversity2(ubcf_topN2, as.matrix(item_sim2))
div_svd2  <- calc_diversity2(svd_topN2,  as.matrix(item_sim2))
div_reg2  <- calc_diversity2(reg_topN2,  as.matrix(item_sim2))

pop_threshold2 <- quantile(item_counts2, 0.9)
unpopular_items2 <- names(item_counts2[item_counts2 < pop_threshold2])
calc_serendipity2 <- function(topNList, actual, unpopular_items) {
  rec_list <- as(topNList, "list")
  actual_list <- as(actual, "list")
  sers <- mapply(function(rec, act) {
    sum(rec %in% act & rec %in% unpopular_items) / length(rec)
  }, rec_list, actual_list)
  mean(sers, na.rm = TRUE)
}
ser_ubcf2 <- calc_serendipity2(ubcf_topN2, actual_unknown2, unpopular_items2)
ser_svd2  <- calc_serendipity2(svd_topN2,  actual_unknown2, unpopular_items2)
ser_reg2  <- calc_serendipity2(reg_topN2,  actual_unknown2, unpopular_items2)
```

```
metrics_df2 <- data.frame(
  Model = c("User-Based CF", "SVD", "Popularity"),
  Precision = c(pr_ubcf2["Precision"], pr_svd2["Precision"], pr_reg2["Precision"]),
  Recall    = c(pr_ubcf2["Recall"],    pr_svd2["Recall"],    pr_reg2["Recall"]),
  Novelty   = c(nov_ubcf2, nov_svd2, nov_reg2),
  Diversity = c(div_ubcf2, div_svd2, div_reg2),
  Serendipity = c(ser_ubcf2, ser_svd2, ser_reg2)
)
knitr::kable(metrics_df2, digits = 4, caption = "Revised Offline Metrics: Precision, Recall, Novelty, D
```

Table 4: Revised Offline Metrics: Precision, Recall, Novelty, Diversity, Serendipity

Model	Precision	Recall	Novelty	Diversity	Serendipity
User-Based CF	0	0	0.8613	0	0
SVD	0	0	0.9973	0	0

Model	Precision	Recall	Novelty	Diversity	Serendipity
Popularity	0	0	0.2956	0	0

Diagnostics: Why Are Metrics Still Zero?

```
ubcf_list <- as(ubcf_topN2, "list")
svd_list  <- as(svd_topN2, "list")
reg_list  <- as(reg_topN2, "list")
cat("UBCF: Proportion with no recommendations:", mean(sapply(ubcf_list, length) == 0), "\n")
```

```
## UBCF: Proportion with no recommendations: 0.01058201
```

```
cat("SVD:  Proportion with no recommendations:", mean(sapply(svd_list,  length) == 0), "\n")
```

```
## SVD:  Proportion with no recommendations: 0.01058201
```

```
cat("Popularity: Proportion with no recommendations:", mean(sapply(reg_list,  length) == 0), "\n")
```

```
## Popularity: Proportion with no recommendations: 0.01058201
```

```
actual_list2 <- as(actual_unknown2, "list")
cat("Proportion with empty test sets:", mean(sapply(actual_list2, length) == 0), "\n")
```

```
## Proportion with empty test sets: 0
```

```
cat("UBCF user with a hit: ", any(sapply(seq_along(ubcf_list), function(i) length(intersect(ubcf_list[[i
```

```
## UBCF user with a hit:  FALSE
```

```
cat("SVD user with a hit: ", any(sapply(seq_along(svd_list), function(i) length(intersect(svd_list[[i]]
```

```
## SVD user with a hit:  FALSE
```

```
cat("Popularity user with a hit: ", any(sapply(seq_along(reg_list), function(i) length(intersect(reg_lis
```

```
## Popularity user with a hit:  FALSE
```

Diagnostics Summary:

Almost all users receive recommendation lists and have non-empty test sets, but none of the recommended items overlap with test items for any user. This reflects a core challenge in sparse recommender datasets—offline evaluation may yield zero hits even for reasonable algorithms.

7. Further Experiment: Looser Evaluation

I experimented with even looser evaluation—using only 3 known ratings per user, recommending 20 items, and leave-one-out splits.

```
set.seed(123)
scheme_loose <- evaluationScheme(MovieLense, method = "cross-validation", k = 5, given = 3, goodRating =
top_n_loose <- 20
ubcf_model_loose <- Recommender(getData(scheme_loose, "train"), method = "UBCF")
svd_model_loose  <- Recommender(getData(scheme_loose, "train"), method = "SVD", parameter = list(k = 20)
reg_model_loose  <- Recommender(getData(scheme_loose, "train"), method = "POPULAR")
ubcf_topN_loose <- predict(ubcf_model_loose, getData(scheme_loose, "known"), type = "topNList", n = top_
svd_topN_loose  <- predict(svd_model_loose,  getData(scheme_loose, "known"), type = "topNList", n = top_
reg_topN_loose  <- predict(reg_model_loose,  getData(scheme_loose, "known"), type = "topNList", n = top_
actual_unknown_loose <- getData(scheme_loose, "unknown")
```

```r
set.seed(123)
scheme_loo <- evaluationScheme(MovieLense, method = "cross-validation", k = 5, given = -1, goodRating =
top_n_loo <- 20
ubcf_model_loo <- Recommender(getData(scheme_loo, "train"), method = "UBCF")
svd_model_loo  <- Recommender(getData(scheme_loo, "train"), method = "SVD", parameter = list(k = 20))
reg_model_loo  <- Recommender(getData(scheme_loo, "train"), method = "POPULAR")
ubcf_topN_loo <- predict(ubcf_model_loo, getData(scheme_loo, "known"), type = "topNList", n = top_n_loo
svd_topN_loo  <- predict(svd_model_loo,  getData(scheme_loo, "known"), type = "topNList", n = top_n_loo
reg_topN_loo  <- predict(reg_model_loo,  getData(scheme_loo, "known"), type = "topNList", n = top_n_loo
actual_unknown_loo <- getData(scheme_loo, "unknown")

precision_recall_exp <- function(pred, actual, n) {
  pred_list <- as(pred, "list")
  actual_list <- as(actual, "list")
  precisions <- mapply(function(p, a) if(length(p)) length(intersect(p, a))/n else NA, pred_list, actual
  recalls    <- mapply(function(p, a) if(length(a)) length(intersect(p, a))/length(a) else NA, pred_lis
  c(Precision = mean(precisions, na.rm = TRUE), Recall = mean(recalls, na.rm = TRUE))
}
pr_ubcf_loose <- precision_recall_exp(ubcf_topN_loose, actual_unknown_loose, top_n_loose)
pr_svd_loose  <- precision_recall_exp(svd_topN_loose,  actual_unknown_loose, top_n_loose)
pr_reg_loose  <- precision_recall_exp(reg_topN_loose,  actual_unknown_loose, top_n_loose)
pr_ubcf_loo <- precision_recall_exp(ubcf_topN_loo, actual_unknown_loo, top_n_loo)
pr_svd_loo  <- precision_recall_exp(svd_topN_loo,  actual_unknown_loo, top_n_loo)
pr_reg_loo  <- precision_recall_exp(reg_topN_loo,  actual_unknown_loo, top_n_loo)

metrics_exp_df <- data.frame(
  Experiment = rep(c("Loose Split (given=3, topN=20)", "Leave-One-Out (topN=20)"), each=3),
  Model = rep(c("UBCF", "SVD", "Popularity"), 2),
  Precision = c(pr_ubcf_loose["Precision"], pr_svd_loose["Precision"], pr_reg_loose["Precision"],
                pr_ubcf_loo["Precision"], pr_svd_loo["Precision"], pr_reg_loo["Precision"]),
  Recall = c(pr_ubcf_loose["Recall"], pr_svd_loose["Recall"], pr_reg_loose["Recall"],
             pr_ubcf_loo["Recall"], pr_svd_loo["Recall"], pr_reg_loo["Recall"])
)
knitr::kable(metrics_exp_df, digits = 4, caption = "Precision and Recall for Looser Evaluation")
```

Table 5: Precision and Recall for Looser Evaluation

Experiment	Model	Precision	Recall
Loose Split (given=3, topN=20)	UBCF	0	0
Loose Split (given=3, topN=20)	SVD	0	0
Loose Split (given=3, topN=20)	Popularity	0	0
Leave-One-Out (topN=20)	UBCF	0	0
Leave-One-Out (topN=20)	SVD	0	0
Leave-One-Out (topN=20)	Popularity	0	0

Result:

Even with looser splits and more recommendations, precision and recall remain zero for all models. This emphasizes the difficulty of offline evaluation with sparse data—most test items are never recommended, and recommended items are rarely held out for testing.

8. Conclusions and Lessons Learned

- **Zero scores for precision/recall/diversity are a natural consequence of data sparsity, not necessarily model failure.**
- **Offline evaluation can be misleadingly pessimistic in sparse settings.**
- **For meaningful comparison, consider denser datasets, alternative splits, or online experiments.**
- **Diversity and novelty are important, but only if recommendations are also relevant.**

This project demonstrates the importance of evaluation design and the challenges of working with real-world, sparse recommender system data.

References

- "Building a Recommendation System in R", Chapters 4–5
- Gunawardana, A., & Shani, G. (2009). A Survey of Accuracy Evaluation Metrics of Recommendation Tasks.
- MovieLens Dataset (link)