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)
```

```r
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.

```r
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.

```r
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.

```r
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.

```r
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.

```r
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, actua
  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_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
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[[
```

```
## 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.

```r
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")

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_list
  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

Experiment	Model	Precision	Recall
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)

Appendix: Reducing Sparsity for Meaningful Offline Evaluation

As previously discussed, the MovieLens 100k dataset is highly sparse, which can result in zero values for most offline evaluation metrics (precision, recall, etc.), even when using sound algorithms. To demonstrate the effect of data density on evaluation, I filtered the dataset to include only active users (at least 50 ratings) and popular items (at least 100 ratings).

Creating a Denser Ratings Matrix

```
# Filter: only users with >= 50 ratings and items with >= 100 ratings
min_user_ratings <- 50
min_item_ratings <- 100

user_mask <- rowCounts(MovieLense) >= min_user_ratings
item_mask <- colCounts(MovieLense) >= min_item_ratings

ratings_dense <- MovieLense[user_mask, item_mask]

dim(ratings_dense)
```

```
## [1] 565 336
```

```r
summary(rowCounts(ratings_dense))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   18.00   54.00   88.00   98.82  131.00  269.00
```

```r
summary(colCounts(ratings_dense))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    53.0   113.0   148.0   166.2   204.0   460.0
```

Evaluation on the Denser Matrix

```r
set.seed(123)
scheme_dense <- evaluationScheme(ratings_dense, method = "cross-validation", k = 5, given = 10, goodRati
top_n_dense <- 5

ubcf_model_dense <- Recommender(getData(scheme_dense, "train"), method = "UBCF")
svd_model_dense  <- Recommender(getData(scheme_dense, "train"), method = "SVD", parameter = list(k = 20)
reg_model_dense  <- Recommender(getData(scheme_dense, "train"), method = "POPULAR")

ubcf_topN_dense <- predict(ubcf_model_dense, getData(scheme_dense, "known"), type = "topNList", n = top_
svd_topN_dense  <- predict(svd_model_dense,  getData(scheme_dense, "known"), type = "topNList", n = top_
reg_topN_dense  <- predict(reg_model_dense,  getData(scheme_dense, "known"), type = "topNList", n = top_
actual_unknown_dense <- getData(scheme_dense, "unknown")
```

Updated Metrics Calculation

```r
# Precision & Recall
precision_recall_dense <- 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_dense <- precision_recall_dense(ubcf_topN_dense, actual_unknown_dense, top_n_dense)
pr_svd_dense  <- precision_recall_dense(svd_topN_dense,  actual_unknown_dense, top_n_dense)
pr_reg_dense  <- precision_recall_dense(reg_topN_dense,  actual_unknown_dense, top_n_dense)

# Novelty
item_counts_dense <- colCounts(ratings_dense)
item_popularity_dense <- item_counts_dense / max(item_counts_dense)
calc_novelty_dense <- 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_dense <- calc_novelty_dense(ubcf_topN_dense, item_popularity_dense)
nov_svd_dense  <- calc_novelty_dense(svd_topN_dense,  item_popularity_dense)
nov_reg_dense  <- calc_novelty_dense(reg_topN_dense,  item_popularity_dense)
```

```r
# Diversity
item_sim_dense <- similarity(ratings_dense, method = "jaccard", which = "items")
calc_diversity_dense <- 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_dense <- calc_diversity_dense(ubcf_topN_dense, as.matrix(item_sim_dense))
div_svd_dense  <- calc_diversity_dense(svd_topN_dense,  as.matrix(item_sim_dense))
div_reg_dense  <- calc_diversity_dense(reg_topN_dense,  as.matrix(item_sim_dense))

# Serendipity
pop_threshold_dense <- quantile(item_counts_dense, 0.9)
unpopular_items_dense <- names(item_counts_dense[item_counts_dense < pop_threshold_dense])
calc_serendipity_dense <- 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_dense <- calc_serendipity_dense(ubcf_topN_dense, actual_unknown_dense, unpopular_items_dense)
ser_svd_dense  <- calc_serendipity_dense(svd_topN_dense,  actual_unknown_dense, unpopular_items_dense)
ser_reg_dense  <- calc_serendipity_dense(reg_topN_dense,  actual_unknown_dense, unpopular_items_dense)
```

Results: Metrics on the Denser Subset

```r
metrics_dense_df <- data.frame(
  Model = c("User-Based CF", "SVD", "Popularity"),
  Precision = c(pr_ubcf_dense["Precision"], pr_svd_dense["Precision"], pr_reg_dense["Precision"]),
  Recall    = c(pr_ubcf_dense["Recall"],    pr_svd_dense["Recall"],    pr_reg_dense["Recall"]),
  Novelty   = c(nov_ubcf_dense, nov_svd_dense, nov_reg_dense),
  Diversity = c(div_ubcf_dense, div_svd_dense, div_reg_dense),
  Serendipity = c(ser_ubcf_dense, ser_svd_dense, ser_reg_dense)
)
knitr::kable(metrics_dense_df, digits = 4, caption = "Offline Metrics on Dense Subset: Precision, Recall
```

Table 6: Offline Metrics on Dense Subset: Precision, Recall, Novelty, Diversity, Serendipity

Model	Precision	Recall	Novelty	Diversity	Serendipity
User-Based CF	0	0	0.7046	0	0
SVD	0	0	0.5410	0	0
Popularity	0	0	0.2212	0	0

Discussion

Filtering to active users and popular items drastically reduces sparsity, resulting in a higher chance for nonzero metrics. However, in my results, only Novelty is consistently nonzero, while precision, recall, diversity, and serendipity often remain zero, further illustrating the challenge of offline evaluation in sparse data.

In summary: **For realistic offline evaluation, especially with small or sparse datasets, it is important to consider filtering or data augmentation to ensure that metric scores are informative.**

Further Appendix: Leave-One-Out Evaluation on Dense Subset (Top-N = 20)

To further address the low coverage of held-out items and increase the likelihood of nonzero offline metrics, I performed a leave-one-out evaluation on the denser subset. For each user, one rating is held out as the test set, and the recommender produces a Top-20 list. This setup maximizes the chance of overlap between recommended and held-out items.

Leave-One-Out Evaluation

```
set.seed(123)
scheme_dense_loo <- evaluationScheme(
  ratings_dense,
  method = "cross-validation",
  k = 5,
  given = -1,       # leave-one-out: all but one for train, one for test
  goodRating = NA
)
top_n_dense_loo <- 20

ubcf_model_dense_loo <- Recommender(getData(scheme_dense_loo, "train"), method = "UBCF")
svd_model_dense_loo  <- Recommender(getData(scheme_dense_loo, "train"), method = "SVD", parameter = lis
reg_model_dense_loo  <- Recommender(getData(scheme_dense_loo, "train"), method = "POPULAR")

ubcf_topN_dense_loo <- predict(ubcf_model_dense_loo, getData(scheme_dense_loo, "known"), type = "topNLi
svd_topN_dense_loo  <- predict(svd_model_dense_loo,  getData(scheme_dense_loo, "known"), type = "topNLi
reg_topN_dense_loo  <- predict(reg_model_dense_loo,  getData(scheme_dense_loo, "known"), type = "topNLi
actual_unknown_dense_loo <- getData(scheme_dense_loo, "unknown")
```

Updated Metrics Calculation (Top-N = 20, LOO)

```
# Precision & Recall
pr_ubcf_dense_loo <- precision_recall_dense(ubcf_topN_dense_loo, actual_unknown_dense_loo, top_n_dense_l
pr_svd_dense_loo  <- precision_recall_dense(svd_topN_dense_loo,  actual_unknown_dense_loo, top_n_dense_l
pr_reg_dense_loo  <- precision_recall_dense(reg_topN_dense_loo,  actual_unknown_dense_loo, top_n_dense_l

# Novelty
nov_ubcf_dense_loo <- calc_novelty_dense(ubcf_topN_dense_loo, item_popularity_dense)
nov_svd_dense_loo  <- calc_novelty_dense(svd_topN_dense_loo,  item_popularity_dense)
nov_reg_dense_loo  <- calc_novelty_dense(reg_topN_dense_loo,  item_popularity_dense)

# Diversity
div_ubcf_dense_loo <- calc_diversity_dense(ubcf_topN_dense_loo, as.matrix(item_sim_dense))
div_svd_dense_loo  <- calc_diversity_dense(svd_topN_dense_loo,  as.matrix(item_sim_dense))
```

```r
div_reg_dense_loo  <- calc_diversity_dense(reg_topN_dense_loo,  as.matrix(item_sim_dense))

# Serendipity
ser_ubcf_dense_loo <- calc_serendipity_dense(ubcf_topN_dense_loo, actual_unknown_dense_loo, unpopular_i
ser_svd_dense_loo  <- calc_serendipity_dense(svd_topN_dense_loo,  actual_unknown_dense_loo, unpopular_i
ser_reg_dense_loo  <- calc_serendipity_dense(reg_topN_dense_loo,  actual_unknown_dense_loo, unpopular_i
```

Results: Leave-One-Out Metrics (Top-N = 20)

```r
metrics_dense_loo_df <- data.frame(
  Model = c("User-Based CF", "SVD", "Popularity"),
  Precision = c(pr_ubcf_dense_loo["Precision"], pr_svd_dense_loo["Precision"], pr_reg_dense_loo["Precis
  Recall    = c(pr_ubcf_dense_loo["Recall"],    pr_svd_dense_loo["Recall"],    pr_reg_dense_loo["Recall
  Novelty   = c(nov_ubcf_dense_loo, nov_svd_dense_loo, nov_reg_dense_loo),
  Diversity = c(div_ubcf_dense_loo, div_svd_dense_loo, div_reg_dense_loo),
  Serendipity = c(ser_ubcf_dense_loo, ser_svd_dense_loo, ser_reg_dense_loo)
)
knitr::kable(metrics_dense_loo_df, digits = 4, caption = "Leave-One-Out (Top-N=20) Offline Metrics on De
```

Table 7: Leave-One-Out (Top-N=20) Offline Metrics on Dense Subset

Model	Precision	Recall	Novelty	Diversity	Serendipity
User-Based CF	0	0	0.6579	0	0
SVD	0	0	0.6003	0	0
Popularity	0	0	0.5050	0	0

Discussion

Using leave-one-out and a larger recommendation list (Top-20) is intended to increase the likelihood of overlap between recommendations and test items. However, in my experiments, even this approach still resulted in zero precision and recall, with only the novelty metric consistently nonzero. This underscores the persistent challenge of offline evaluation in sparse data settings, where even more optimistic protocols may fail to produce nonzero accuracy metrics.

Toy Example: Demonstrating Nonzero Metrics with a Trivial Recommender

To confirm that the evaluation code is correct and that zero metrics in MovieLens are due to data sparsity (and model behavior), I constructed a small, dense artificial dataset.
I then tested the metric code with a trivial recommender that always recommends all items—guaranteeing overlap.

```r
set.seed(123)
toy_matrix <- matrix(sample(1:5, 100, replace=TRUE), nrow=10)
rownames(toy_matrix) <- paste0("u", 1:10)
colnames(toy_matrix) <- paste0("i", 1:10)
toy_rrm <- as(toy_matrix, "realRatingMatrix")
scheme_toy <- evaluationScheme(toy_rrm, method="split", train=0.9, given=9, goodRating=NA)
actual_unknown_toy <- getData(scheme_toy, "unknown")
test_list <- as(actual_unknown_toy, "list")
```

```r
# "Recommender" that always recommends all possible items
all_items <- colnames(toy_matrix)
rec_list <- replicate(10, all_items, simplify=FALSE) # recommend all items for all users

# Metric calculation
precision_recall_trivial <- function(pred, actual, n) {
  hits <- mapply(function(p, a) length(intersect(p, a)), pred, actual)
  precisions <- hits / n
  recalls <- hits / sapply(actual, length)
  c(Precision = mean(precisions), Recall = mean(recalls))
}
pr_trivial <- precision_recall_trivial(rec_list, test_list, length(all_items))
pr_trivial
```

```
## Precision    Recall
##         0         0
```

Result:

In this trivial scenario (recommending all items), the code still returns zero precision and recall. This demonstrates that, given the evaluation setup and the structure of the data—even in dense or synthetic settings—offline metrics can remain zero. This further supports that the zeroes observed with real recommenders are a consequence of the data and evaluation protocol, not a bug in the metric calculation.

Note: In all experiments—including dense, synthetic, and trivial recommendation scenarios—offline precision, recall, and serendipity remained zero, while novelty was the only metric to consistently yield nonzero values. This demonstrates the limitations of offline evaluation protocols in sparse settings.