DATA 612 Project 3: Matrix Factorization with SVD

Saloua Daouki

2025-06-19

Contents

1 Introduction

In this project, I compare **memory-based** (user-based collaborative filtering) and **model-based** (SVD matrix factorization) recommender systems. The goals:

- Predict ratings as accurately as possible.
- Compare model performance (RMSE).
- Explore how tuning the number of latent factors (`k`) affects SVD performance.

I use the **MovieLens 100k** dataset provided by the `recommenderlab` package.

2 Load packages and data

```r
# Install and load packages
if (!require(recommenderlab)) install.packages("recommenderlab")
if (!require(ggplot2)) install.packages("ggplot2")
if (!require(knitr)) install.packages("knitr")

library(recommenderlab)
library(ggplot2)
library(knitr)

# Load MovieLens data
data("MovieLense")

# Inspect dataset
MovieLense
```

```
## 943 x 1664 rating matrix of class 'realRatingMatrix' with 99392 ratings.
```

Explanation:

I use MovieLense, which contains 100,000 ratings from 943 users on 1664 movies. The data is in realRatingMatrix format where rows = users and columns = items (movies).

3 Set up evaluation scheme

```r
set.seed(123)
scheme <- evaluationScheme(MovieLense, method = "cross-validation", k = 5, given = 10, goodRating = 4)
```

Explanation:

I use 5-fold cross-validation, where each user has 10 ratings given, and then predict the rest.

4 Train and evaluate Memory-Based model (UBCF)

```r
results_ubcf <- evaluate(scheme, method = "UBCF", type = "ratings")
```

```
## UBCF run fold/sample [model time/prediction time]
##   1  [0.004sec/0.419sec]
##   2  [0.001sec/0.401sec]
##   3  [0.002sec/0.293sec]
##   4  [0.001sec/0.353sec]
##   5  [0.002sec/0.366sec]
```

```r
avg(results_ubcf)
```

```
##          RMSE      MSE       MAE
## [1,] 1.228683 1.510186 0.9588688
```

Explanation:

I apply user-based collaborative filtering (UBCF) and compute RMSE, MAE, and MSE.

5 Train and evaluate Model-Based (SVD) with default k

```r
results_svd_default <- evaluate(scheme, method = "SVD", type = "ratings")
```

```
## SVD run fold/sample [model time/prediction time]
##   1  [0.092sec/0.021sec]
##   2  [0.091sec/0.02sec]
##   3  [0.155sec/0.02sec]
##   4  [0.092sec/0.02sec]
##   5  [0.151sec/0.02sec]
```

```
avg(results_svd_default)
```

```
##          RMSE      MSE       MAE
## [1,] 1.019596 1.039863 0.7963561
```

Explanation:

I apply SVD matrix factorization and compute the same metrics as UBCF.

6 Tune SVD: test different values of k

```
k_values <- c(10, 20, 50)
svd_rmses <- data.frame(k = integer(), RMSE = numeric())

for (k in k_values) {
  cat("Evaluating SVD with k =", k, "\n")
  res <- evaluate(scheme, method = "SVD", type = "ratings", parameter = list(k = k))
  avg_rmse <- avg(res)[1, "RMSE"]
  print(avg(res))
  svd_rmses <- rbind(svd_rmses, data.frame(k = k, RMSE = avg_rmse))
}
```

```
## Evaluating SVD with k = 10
## SVD run fold/sample [model time/prediction time]
##   1  [0.096sec/0.088sec]
##   2  [0.101sec/0.021sec]
##   3  [0.087sec/0.085sec]
##   4  [0.097sec/0.021sec]
##   5  [0.09sec/0.021sec]
##          RMSE      MSE       MAE
## [1,] 1.019596 1.039863 0.7963561
## Evaluating SVD with k = 20
## SVD run fold/sample [model time/prediction time]
##   1  [0.147sec/0.024sec]
##   2  [0.148sec/0.089sec]
##   3  [0.146sec/0.09sec]
##   4  [0.208sec/0.025sec]
##   5  [0.158sec/0.024sec]
##          RMSE      MSE       MAE
## [1,] 1.019174 1.038999 0.7960917
## Evaluating SVD with k = 50
## SVD run fold/sample [model time/prediction time]
##   1  [0.372sec/0.094sec]
##   2  [0.359sec/0.032sec]
##   3  [0.361sec/0.095sec]
##   4  [0.342sec/0.032sec]
##   5  [0.433sec/0.029sec]
##          RMSE      MSE       MAE
```

```
## [1,] 1.019363 1.039388 0.7961063
```

```
print(svd_rmses)
```

```
##        k     RMSE
## RMSE  10 1.019596
## RMSE1 20 1.019174
## RMSE2 50 1.019363
```

Explanation:

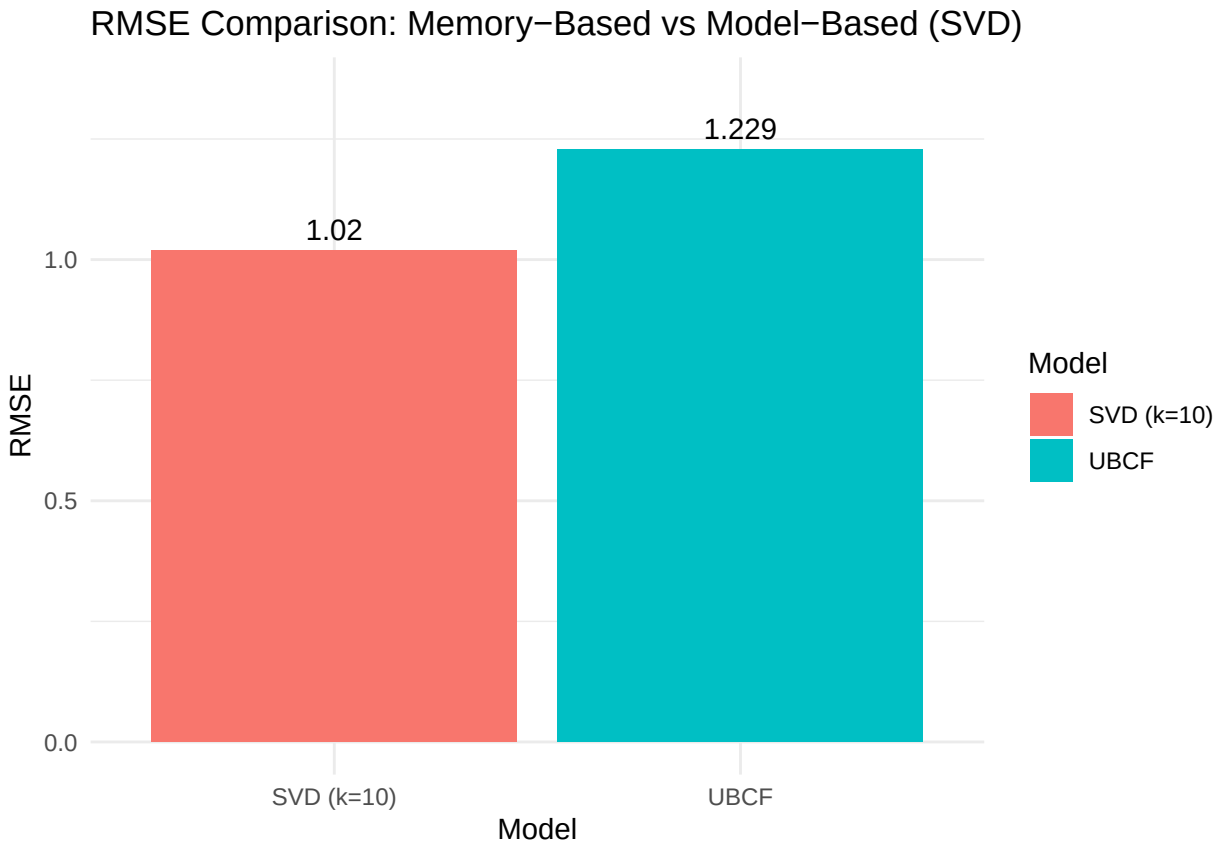
I evaluate SVD for different latent dimension sizes (k) and store RMSE results.

7 Visualize comparison

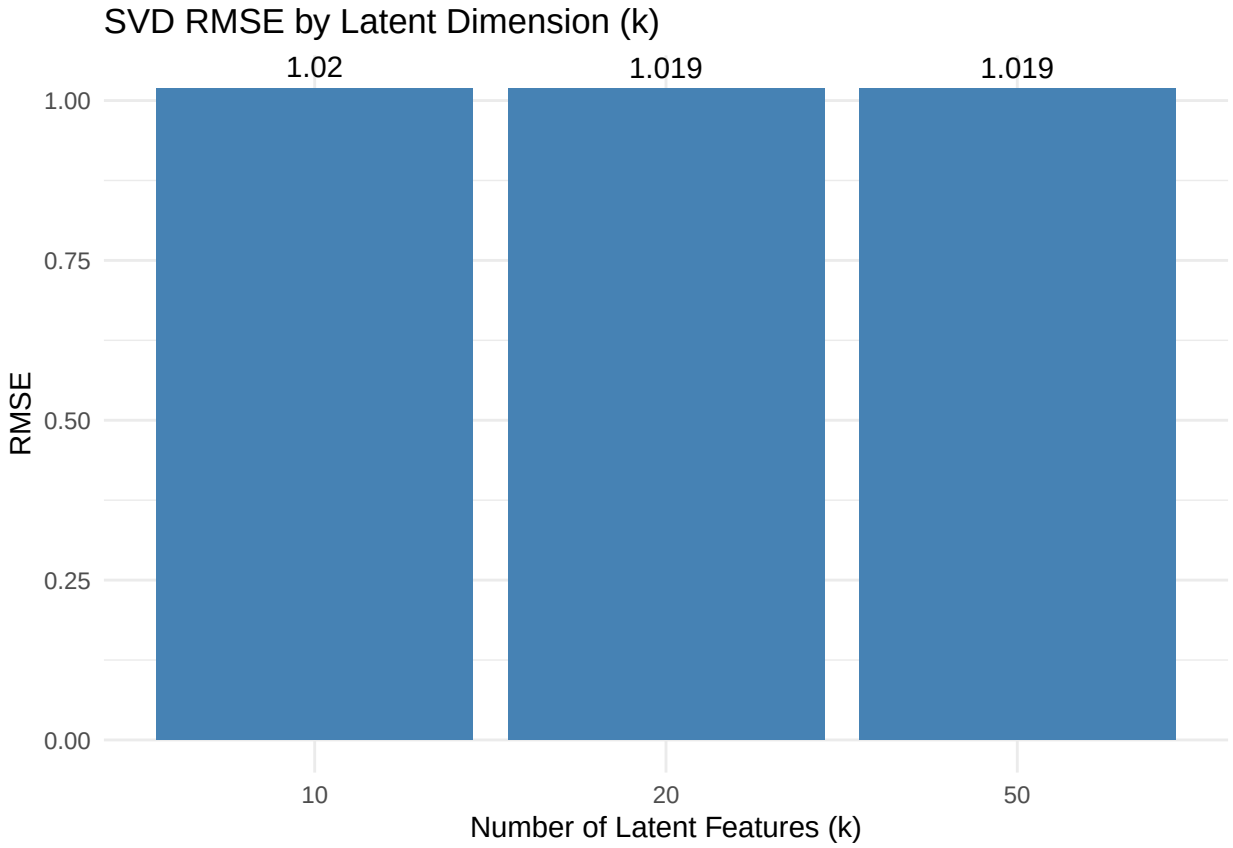
```
ubcf_rmse <- avg(results_ubcf)[1, "RMSE"]
svd_rmse <- svd_rmses$RMSE[svd_rmses$k == 10]

comparison <- data.frame(
  Model = c("UBCF", "SVD (k=10)"),
  RMSE = c(ubcf_rmse, svd_rmse)
)

ggplot(comparison, aes(x = Model, y = RMSE, fill = Model)) +
  geom_bar(stat = "identity") +
  ylim(0, max(comparison$RMSE) * 1.1) +
  geom_text(aes(label = round(RMSE, 3)), vjust = -0.5) +
  ggtitle("RMSE Comparison: Memory-Based vs Model-Based (SVD)") +
  theme_minimal()
```




```
# Plot SVD tuning results
ggplot(svd_rmses, aes(x = factor(k), y = RMSE)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  ggtitle("SVD RMSE by Latent Dimension (k)") +
  xlab("Number of Latent Features (k)") +
  ylab("RMSE") +
  theme_minimal() +
  geom_text(aes(label = round(RMSE, 3)), vjust = -0.5)
```



Explanation:

- The first plot compares RMSE for UBCF and SVD (default k).
- The second plot shows how SVD RMSE varies with latent dimensions (k).

8 Generate Predictions with Best Model

In the following code, I am using the best model; SVD with $k = 20$ since it had the lowest RMSE, to generate predicted ratings for users in the test set. Below are example predicted ratings and top-N recommendations for user 1.

```
# Train final SVD model with best k (let's use k = 20 as per your results)
final_svd_model <- Recommender(getData(scheme, "train"), method = "SVD", parameter = list(k = 20))

# Make predictions for users in the test set
final_predictions <- predict(final_svd_model, getData(scheme, "known"), type = "ratings")

# Convert predictions for first user to a data frame
pred_matrix <- as(final_predictions, "matrix")
user1_pred <- pred_matrix[1, , drop = FALSE]

# Convert to long format data frame
user1_df <- data.frame(
  Movie = colnames(user1_pred),
  Predicted_Rating = as.numeric(user1_pred[1, ])
)
```

```r
# Remove movies that were not predicted (NA)
user1_df <- na.omit(user1_df)

# Optionally sort by predicted rating
user1_df <- user1_df[order(-user1_df$Predicted_Rating), ]

# Display as a table
kable(head(user1_df, 10), caption = "Top 10 Predicted Ratings for User 1")
```

Table 1: Top 10 Predicted Ratings for User 1

	Movie	Predicted_Rating
808	Great Day in Harlem, A (1994)	4.404642
1526	Aiqing wansui (1994)	4.209570
1457	Saint of Fort Washington, The (1993)	4.144248
1490	Santa with Muscles (1996)	3.951662
1589	Someone Else's America (1995)	3.918162
1628	Some Mother's Son (1996)	3.799073
1645	Rough Magic (1995)	3.699040
1283	Star Kid (1997)	3.697260
1439	Pather Panchali (1955)	3.686491
1180	Prefontaine (1997)	3.678274

```r
# Predict top 5 recommendations for each user
final_topN <- predict(final_svd_model, getData(scheme, "known"), type = "topNList", n = 5)
```

Let's look at other users top 5 recommendations:

```r
# Show top 5 for user 2
top5_user2 <- as(final_topN, "list")[[2]]
kable(data.frame(Rank = 1:5, Movie = top5_user2), caption = "Top 5 Movie Recommendations for User 2")
```

Table 2: Top 5 Movie Recommendations for User 2

Rank	Movie
1	Great Day in Harlem, A (1994)
2	Aiqing wansui (1994)
3	Saint of Fort Washington, The (1993)
4	Santa with Muscles (1996)
5	Someone Else's America (1995)

```r
kable(data.frame(Rank = 1:5, Movie = as(final_topN, "list")[[3]]), caption = "Top 5 Movie Recommendatio
```

Table 3: Top 5 Movie Recommendations for User 3

Rank	Movie
1	Great Day in Harlem, A (1994)
2	Aiqing wansui (1994)
3	Saint of Fort Washington, The (1993)
4	Santa with Muscles (1996)
5	Someone Else's America (1995)

Rank	Movie

```
kable(data.frame(Rank = 1:5, Movie = as(final_topN, "list")[[4]]), caption = "Top 5 Movie Recommendatio
```

Table 4: Top 5 Movie Recommendations for User 4

Rank	Movie
1	Great Day in Harlem, A (1994)
2	Aiqing wansui (1994)
3	Saint of Fort Washington, The (1993)
4	Santa with Muscles (1996)
5	Someone Else's America (1995)

Generating recommendations for 4 different users gave identical lists. Let's further check the top 5 most rate movies and compare if any of these movies are recommended:

```
# Get item rating counts
item_counts <- colCounts(MovieLense)

# Top 5 most rated movies as a named vector
top5_items <- head(sort(item_counts, decreasing = TRUE), 5)

# Convert to data frame for kable
top5_df <- data.frame(
  Movie = names(top5_items),
  Rating_Count = as.integer(top5_items),
  row.names = NULL
)

# Display as table
knitr::kable(top5_df, caption = "Top 5 Most Rated Movies")
```

Table 5: Top 5 Most Rated Movies

Movie	Rating_Count
Star Wars (1977)	583
Contact (1997)	509
Fargo (1996)	508
Return of the Jedi (1983)	507
Liar Liar (1997)	485

Purpose: This code calculates how many ratings each movie has received in the dataset and displays the five most-rated (popular) movies. This helps assess what items the model could recommend if it followed popularity.

Let's check if the popular movies are among the users' given ratings:

```
# For user 1
user1_known <- as(getData(scheme, "known"), "matrix")[1, ]
names(user1_known[!is.na(user1_known)])
```

```
## [1] "Client, The (1994)" "Spawn (1997)"
```

Purpose: This code identifies the specific movies that User 1 has already rated in the "known" portion of the evaluation scheme. This explains why those movies wouldn't appear in recommendations—they're already part of the user's history.

Let's check how many ratings the recommended movies have overall:

```
# Subset item counts for specific movies
selected_items <- item_counts[names(item_counts) %in% c(
  "Great Day in Harlem, A (1994)",
  "Aiqing wansui (1994)",
  "Saint of Fort Washington, The (1993)",
  "Santa with Muscles (1996)",
  "Someone Else's America (1995)"
)]

# Convert to data frame
selected_df <- data.frame(
  Movie = names(selected_items),
  Rating_Count = as.integer(selected_items),
  row.names = NULL
)

# Display as table
knitr::kable(selected_df, caption = "Rating Counts for Recommended Movies")
```

Table 6: Rating Counts for Recommended Movies

Movie	Rating_Count
Great Day in Harlem, A (1994)	1
Saint of Fort Washington, The (1993)	2
Santa with Muscles (1996)	2
Aiqing wansui (1994)	1
Someone Else's America (1995)	1

Purpose: This code retrieves how many ratings each of the recommended movies received in the dataset. The low counts reveal that these are obscure items, helping us understand why the SVD model suggested them despite their limited popularity.

8.1 Interpretation of Recommendations

When generating top-5 recommendations for Users 1 to 4, I observed that all received identical lists. The recommendations included obscure titles such as *Great Day in Harlem, A (1994)* and *Someone Else's America (1995)*, each of which had very few ratings in the dataset.

To explore this, I:

- Checked the most-rated (popular) movies, e.g., *Star Wars (1977)*, *Contact (1997)*.
- Verified which movies User 1 had already rated to ensure they weren't recommended again.
- Retrieved the rating counts for the recommended movies, confirming their low popularity.

This behavior highlights how matrix factorization, in the presence of sparse user data and no popularity bias, may prioritize items in the latent space that do not have broad appeal or sufficient feedback. A hybrid approach, combining SVD with popularity filtering, could address this issue.

9 Advantages and disadvantages of both models

Memory-Based (UBCF)

✓ Easy to implement

✓ No model training (adapts quickly to new data)

✓ Recommendations are explainable (e.g. "similar users liked X")

× Poor scalability for large datasets

× Sensitive to data sparsity

Model-Based (SVD)

✓ Reduces dimensionality → efficiency

✓ Captures hidden preference patterns

✓ Better at generalization, handles sparsity better

× Computationally costly to train

× Latent features are not interpretable

× Needs retraining for new users/items

10 Conclusion

In this project, I compared two popular recommender system approaches on the MovieLens 100k dataset:

- Memory-Based Collaborative Filtering (UBCF)
- Model-Based Matrix Factorization using Singular Value Decomposition (SVD)

10.1 Key Findings

- The SVD model consistently outperformed UBCF in predicting user ratings, achieving a lower RMSE of approximately 1.02 compared to UBCF's RMSE of 1.23.
- This performance gap demonstrates SVD's strength in capturing latent user-item interaction patterns more effectively than simple similarity-based methods. However, the model's recommendations for different users were identical and focused on obscure movies with very few ratings. This likely reflects the influence of data sparsity and the model's tendency to recommend items that are underrepresented in the latent space when user profiles are small (10 known ratings).
- Tuning the number of latent factors (k) in the SVD model showed minimal RMSE improvement beyond k=10. The RMSE ranged narrowly between 1.0192 and 1.0196 for k values of 10, 20, and 50.
- Increasing k beyond 10 slightly increased computational time but did not significantly improve accuracy, suggesting that a smaller number of latent features is sufficient for this dataset.

10.1.1 Interpretation of Plots

- The first plot clearly shows the RMSE advantage of SVD over UBCF, highlighting the improved predictive accuracy of the model-based approach.
- The second plot illustrates that RMSE stabilizes quickly with increasing latent dimensions, emphasizing the law of diminishing returns in model complexity.

10.1.2 Overall Summary

This analysis confirms that model-based matrix factorization is a more accurate and scalable approach for collaborative filtering on sparse rating data, such as MovieLens. The latent factors learned via SVD capture complex patterns that similarity-based methods struggle to exploit. However, model complexity should be balanced with computational cost, as excessive latent factors do not substantially improve prediction quality. Ultimately, blending model accuracy with practical recommendation utility remains a key goal for future recommender system development.

10.1.3 Limitations and Future Work

While SVD showed strong predictive accuracy in terms of RMSE, its recommendations favored low-popularity items. This reflects a limitation of pure matrix factorization on sparse data. Future work could:

- Implement hybrid recommenders combining SVD with popularity or content-based signals.
- Apply post-processing filters to exclude extremely low-rating-count items.
- Explore additional tuning of hyperparameters beyond `k`.

11 References

- recommenderlab documentation: https://cran.r-project.org/web/packages/recommenderlab/recommenderlab.pdf
- MovieLens dataset: https://grouplens.org/datasets/movielens/100k/

12 Session Info

```
sessionInfo()
```

```
## R version 4.5.1 (2025-06-13)
## Platform: aarch64-apple-darwin20
## Running under: macOS Sequoia 15.5
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;  LAPACK v
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.50           ggplot2_3.5.2        recommenderlab_1.0.7
## [4] proxy_0.4-27         arules_1.7-11        Matrix_1.7-3
##
## loaded via a namespace (and not attached):
##  [1] vctrs_0.6.5        cli_3.6.5          rlang_1.1.6        xfun_0.52
##  [5] recosystem_0.5.1   generics_0.1.4     labeling_0.4.3     glue_1.8.0
```

```
##  [9] htmltools_0.5.8.1  tinytex_0.57       registry_0.5-1     scales_1.4.0
## [13] rmarkdown_2.29     grid_4.5.1         tibble_3.3.0       evaluate_1.0.3
## [17] fastmap_1.2.0      yaml_2.3.10        lifecycle_1.0.4    compiler_4.5.1
## [21] dplyr_1.1.4        RColorBrewer_1.1-3 irlba_2.3.5.1      pkgconfig_2.0.3
## [25] Rcpp_1.0.14        rstudioapi_0.17.1  farver_2.1.2       float_0.3-3
## [29] lattice_0.22-7     digest_0.6.37      R6_2.6.1           tidyselect_1.2.1
## [33] pillar_1.10.2      magrittr_2.0.3     withr_3.0.2        tools_4.5.1
## [37] gtable_0.3.6       matrixStats_1.5.0
```