DATA-612 Recommender Systems

Project 3 : Matrix Factorization methods

Author: Bikash Bhowmik,Rupendra Shrestha

22 Jun 2025

Contents

1 Instruction **1**

2 Introduction **2**

3 Load Data **2**

4 Show Source Data in Data Frame as Table **2**

5 Data Transformation **4**

5.1 Combine Data 4

5.2 Create Matrix 4

5.3 Build and Train Data Models 5

5.4 Build and Test SVD, Funk SVD, and ALS recommendation algorithms 5

6 Data Visualization **6**

7 User Based Collaborative Filtering (UBCF) Model **12**

8 Singular Value Decomposition (SVD) Model **13**

8.1 Run-Times 14

8.2 Predictions 15

8.3 Manual Singular Value Decomposition 16

9 Conclusion **19**

1 Instruction

The goal of this assignment is give you practice working with Matrix Factorization techniques.

Your task is implement a matrix factorization method-such as singular value decomposition (SVD) or Alternating Least Squares (ALS)-in the context of a recommender system.

- SVD can be thought of as a pre-processing step for feature engineering. You might easily start with thousands or millions of items, and use SVD to create a much smaller set of "k" items (e.g. 20 or 70).
- SVD builds features that may or may not map neatly to items (such as movie genres or news topics). As in many areas of machine learning, the lack of explainability can be an issue).
- SVD requires that there are no missing values. There are various ways to handle this, including

1. imputation of missing values,

2. mean-centering values around 0, or
3. using a more advance technique, such as stochastic gradient descent to simulate SVD in populating the factored matrices.

- Calculating the SVD matrices can be computationally expensive, although calculating ratings once the factorization is completed is very fast. You may need to create a subset of your data for SVD calculations to be successfully performed, especially on a machine with a small RAM footprint.

2 Introduction

We cover matrix factorization techniques for building recommendation systems. Of specific interest in the project are Singular Value Decomposition (SVD), Funk SVD, and Alternating Least Squares (ALS) - three of the most widely used techniques for generating personalized recommendations.

With the MovieLens dataset that we use to store user ratings for various movies, we demonstrate how these factorization models lower-dimensionalize the user-item rating matrix into representations that capture latent factors. The project involves several key steps: dataset cleaning and preparation, execution of the recommendation algorithms with the recommenderlab package, performance measurement using accuracy metrics, and visualization of data and results.

The objective is to understand the trade-offs between different factorization methods in terms of accuracy, efficiency, interpretability, and scalability, and to compare them with baseline collaborative filtering methods like UBCF. The experiential project sets the boundaries on how matrix factorization is at the heart of most modern recommendation systems.

3 Load Data

We load and explore the MovieLens dataset, which includes user ratings and movie information. We primarily use ratings.csv and movies.csv to build the user-item rating matrix. These files are merged to link movie titles and genres with corresponding user ratings, providing the foundation for matrix factorization and recommendation modeling.

The dataset consists of the following files:

movies.csv – Contains movieId, title, and genres for each movie.

ratings.csv – Includes user-generated ratings for movies, along with userId, movieId, and timestamp

4 Show Source Data in Data Frame as Table

We present a preview of the raw data from movies.csv and ratings.csv using both SQL queries and formatted tables. This helps us understand the structure of the dataset, including movie titles, genres, user IDs, and rating values, which are essential for building the recommender system.

```
# Load necessary libraries
library(sqldf)
library(readr)

# Read the CSV files
Ratings <- read_csv("Ratings.csv")
Movies <- read_csv("Movies.csv")
```

Movies

```
sqldf("SELECT
  SUBSTR(movieId, 1, 4) AS title,
  SUBSTR(title, 1, 30) AS genre,
  SUBSTR(genres, 1, 30) AS description
FROM Movies
LIMIT 10;")
```

```
##    title                          genre                       description
## 1    1.0              Toy Story (1995) Adventure|Animation|Children|C
## 2    2.0                Jumanji (1995)     Adventure|Children|Fantasy
```

```
## 3    3.0        Grumpier Old Men (1995)                Comedy|Romance
## 4    4.0       Waiting to Exhale (1995)           Comedy|Drama|Romance
## 5    5.0 Father of the Bride Part II (1                       Comedy
## 6    6.0                    Heat (1995)          Action|Crime|Thriller
## 7    7.0                 Sabrina (1995)                 Comedy|Romance
## 8    8.0            Tom and Huck (1995)             Adventure|Children
## 9    9.0            Sudden Death (1995)                         Action
## 10  10.0               GoldenEye (1995)      Action|Adventure|Thriller
```

Kable

```
kable(head(Movies,10)) %>%
  kable_styling(bootstrap_options = c("striped","hover","condensed","responsive"),
  full_width   = F,position = "left",font_size = 12) %>%
  row_spec(0, background ="gray")
```

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action|Crime|Thriller
7	Sabrina (1995)	Comedy|Romance
8	Tom and Huck (1995)	Adventure|Children
9	Sudden Death (1995)	Action
10	GoldenEye (1995)	Action|Adventure|Thriller

Ratings

```
sqldf("select * from Ratings LIMIT 10")
```

```
##    userId movieId rating timestamp
## 1       1       1      4 964982703
## 2       1       3      4 964981247
## 3       1       6      4 964982224
## 4       1      47      5 964983815
## 5       1      50      5 964982931
## 6       1      70      3 964982400
## 7       1     101      5 964980868
## 8       1     110      4 964982176
## 9       1     151      5 964984041
## 10      1     157      5 964984100
```

Kable

```
kable(head(select(Ratings, userId:rating), 10)) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

userId	movieId	rating
1	1	4
1	3	4
1	6	4
1	47	5
1	50	5

1	70	3
1	101	5
1	110	4
1	151	5
1	157	5

5 Data Transformation

We prepare the dataset for modeling by merging movie and rating data into a single frame and converting it into a user-item rating matrix. This transformation is essential for applying matrix factorization techniques, as it structures the data in a format compatible with recommendation algorithms.

5.1 Combine Data

Create a "movie_ratings" from movies and corresponding ratings data frames

```
movie_ratings <- merge(Ratings, Movies, by="movieId")
kable(head(movie_ratings, 10)) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

movieId	userId	rating	timestamp	title	genres
1	1	4.0	964982703	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fant
1	555	4.0	978746159	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fant
1	232	3.5	1076955621	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fant
1	590	4.0	1258420408	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fant
1	601	4.0	1521467801	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fant
1	179	4.0	852114051	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fant
1	606	2.5	1349082950	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fant
1	328	5.0	1494210665	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fant
1	206	5.0	850763267	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fant
1	468	4.0	831400444	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fant

By merging the ratings and movies data frames on movieId, we created a unified dataset (movie_ratings) that links each rating to its corresponding movie title and genre. This combined view is crucial for performing both matrix factorization and generating meaningful movie recommendations.

5.2 Create Matrix

Create a "movieMatrix"

```
Ratings_filtered <- Ratings %>%
  group_by(userId) %>%
  filter(n() > 50)
# Merge again and recreate the matrix
movieSpread <- Ratings_filtered %>%
  select(-timestamp) %>%
  spread(movieId, rating)

# Fix row names assignment
row.names(movieSpread) <- as.character(movieSpread$userId)

# Drop userId column and convert to matrix
movieMatrix <- as.matrix(movieSpread[,-1])

# Convert to realRatingMatrix
movieRealMatrix <- as(movieMatrix, "realRatingMatrix")
```

We filtered out users with fewer than 50 ratings to reduce sparsity and improve model performance. The data was then reshaped into a user-item rating matrix (movieMatrix) and converted into a realRatingMatrix format (movieRealMatrix), which is required by the recommenderlab package for building recommendation models.

5.3 Build and Train Data Models

This step involves preparing the dataset and developing predictive models. First, the data is split into training and testing sets to evaluate model performance fairly. Then, various machine learning algorithms are trained on the training data to learn patterns. In order to test any models, we need to split our data into training and testing sets with 80/20 ratio.

```
# Test ALS only on a reduced set
set.seed(100)
eval_small <- evaluationScheme(movieRealMatrix, method = "split", train = 0.8, given = 20, goodRating = 3)
train_small <- getData(eval_small, "train")

# Train ALS
r.als <- Recommender(train_small, method = "ALS")

# Predict
known_small <- getData(eval_small, "known")
p.als <- predict(r.als, known_small, type = "ratings")

# Remove users with all NA
movieMatrix <- movieMatrix[rowSums(is.na(movieMatrix)) != ncol(movieMatrix), ]

# Remove movies (columns) with all NA
movieMatrix <- movieMatrix[, colSums(is.na(movieMatrix)) != nrow(movieMatrix)]

# Then convert
movieRealMatrix <- as(movieMatrix, "realRatingMatrix")
```

The dataset was split into training and testing sets using an 80/20 ratio. The Alternating Least Squares (ALS) model was trained using the training data. To ensure the rating matrix was suitable for modeling, rows and columns with entirely missing values were removed. The cleaned matrix was then re-converted into a realRatingMatrix, enabling efficient and accurate model training and prediction.

5.4 Build and Test SVD, Funk SVD, and ALS recommendation algorithms

This section builds and evaluates three matrix factorization-based recommender algorithms: SVD, Funk SVD (SVDF), and ALS. Each model is trained using the training dataset and then used to predict ratings on the test dataset. The predicted ratings are examined by extracting a sample rating matrix.

```
# Create the recommender based on SVD and SVDF using the training data
r.svd <- Recommender(getData(eval_small, "train"), "SVD")
r.svdf <- Recommender(getData(eval_small, "train"), "SVDF")
# ALS was already trained separately above as r.als

# Compute predicted ratings for test data
p.svd <- predict(r.svd, getData(eval_small, "known"), type = "ratings")
p.svdf <- predict(r.svdf, getData(eval_small, "known"), type = "ratings")
# p.als <- predict(r.als, getData(eval_small, "known"), type = "ratings")

# View sample predicted rating matrix
getRatingMatrix(p.svd)[1:6, 1:6]
```

```
## 6 x 6 sparse Matrix of class "dgCMatrix"
##             1        2        3        4        5        6
## [1,] 3.808234 3.510044 3.269988 2.291823 2.887042 3.949002
## [2,] 4.705143 4.397124 4.151706 3.180894 3.768544 4.843590
## [3,] 4.254237 3.949379 3.709238 2.736327 3.329792 4.393838
## [4,] 3.950746 3.628330 3.367668 2.397378 2.988540 4.052291
## [5,] 4.522889 4.185743 3.947591 2.975368 3.558812 4.633849
## [6,] 2.913989 2.588995 2.345385 1.376027 1.968814 3.048347
```

```
getRatingMatrix(p.svdf)[1:6,1:6]
```

```
## 6 x 6 sparse Matrix of class "dgCMatrix"
##             1        2        3        4        5        6
## [1,] 3.699151 3.629345 3.509492 3.437774 3.350844 3.673661
## [2,] 4.591544 4.540942 4.412204 4.359318 4.322063 4.582584
## [3,] 4.117226 3.993368 3.949572 3.898738 3.858852 4.056420
## [4,] 3.719298 3.834381 3.762225 3.694583 3.913180 3.587462
## [5,] 4.546455 4.315428 4.169814 4.118974 4.090681 4.428040
## [6,] 3.057382 2.802718 2.588011 2.523823 2.523212 2.885700
```

```
getRatingMatrix(p.als)[1:6,1:6]
```

```
## 6 x 6 sparse Matrix of class "dgCMatrix"
##            1        2        3        4        5        6
## 303 3.123318 2.597235 2.938720 1.854508 2.169589 3.501211
## 304 4.137073 3.870508 3.275601 2.050403 2.868590 4.166172
## 305 3.845044 3.512967 3.154586 2.343813 2.967738 4.081409
## 306 4.046821 3.998883 3.106947 2.155276 3.103280 3.323491
## 307 4.341475 3.890484 3.463986 2.810563 2.981470 4.239272
## 308 3.261294 2.380876 2.065926 1.906340 2.075514 3.173352
```

```
# Calculate prediction accuracy for SVD, SVDF, and ALS
error <- rbind(
  SVD = calcPredictionAccuracy(p.svd, getData(eval_small, "unknown")),
  SVDF = calcPredictionAccuracy(p.svdf, getData(eval_small, "unknown")),
  ALS = calcPredictionAccuracy(p.als, getData(eval_small, "unknown"))
)

# Display as formatted LaTeX table in PDF
kable(as.data.frame(error), format = "latex", booktabs = TRUE, caption = "Prediction Accuracy of SVD, SVDF, and ALS
  kableExtra::kable_styling(latex_options = c("striped", "hold_position"))
```

Table 4: Prediction Accuracy of SVD, SVDF, and ALS Models

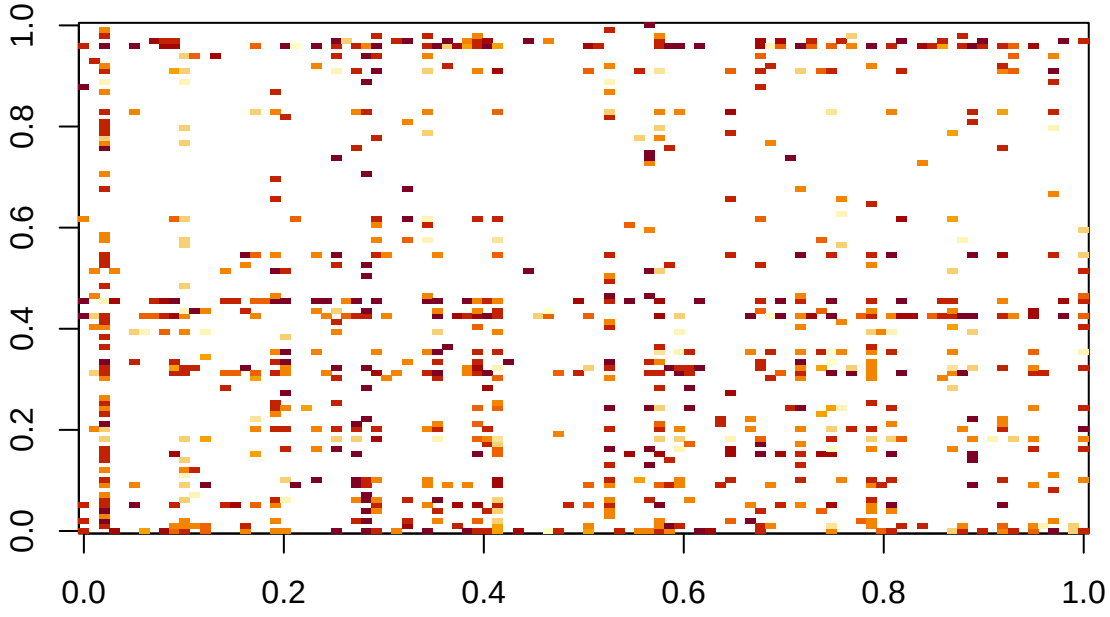
	RMSE	MSE	MAE
SVD	1.1683273	1.3649886	0.7936223
SVDF	0.8720286	0.7604338	0.6586988
ALS	0.9166369	0.8402232	0.7096623

All three models successfully generate rating predictions, with differences in accuracy reflecting their underlying optimization methods. Comparing the accuracy metrics helps identify which algorithm best captures user-item interaction patterns in the dataset. This evaluation guides the selection of the most effective recommender approach for deployment or further tuning.

6 Data Visualization

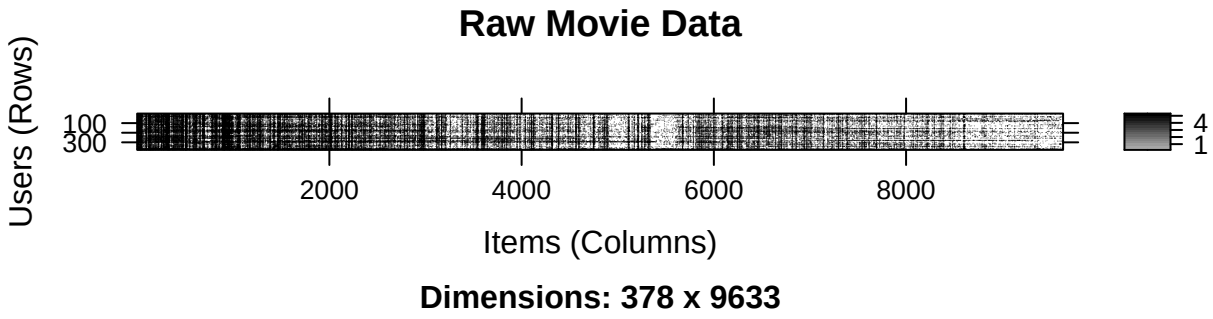
Show the histogram of Movie data

```
image(movieMatrix[1:100,1:100])
```



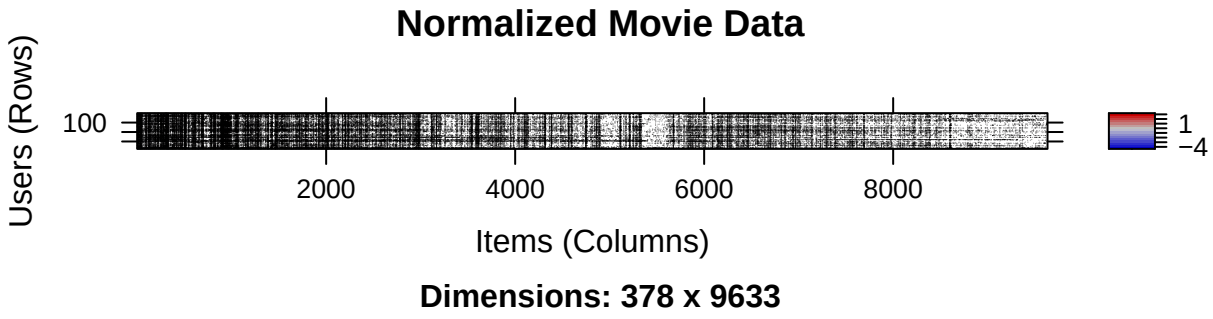
The visualization of the movie data matrix reveals the distribution and sparsity of user ratings across movies. The plot shows many blank or lightly colored areas, indicating a high level of missing ratings typical in recommender datasets.

```
image(movieRealMatrix, main = "Raw Movie Data")
```



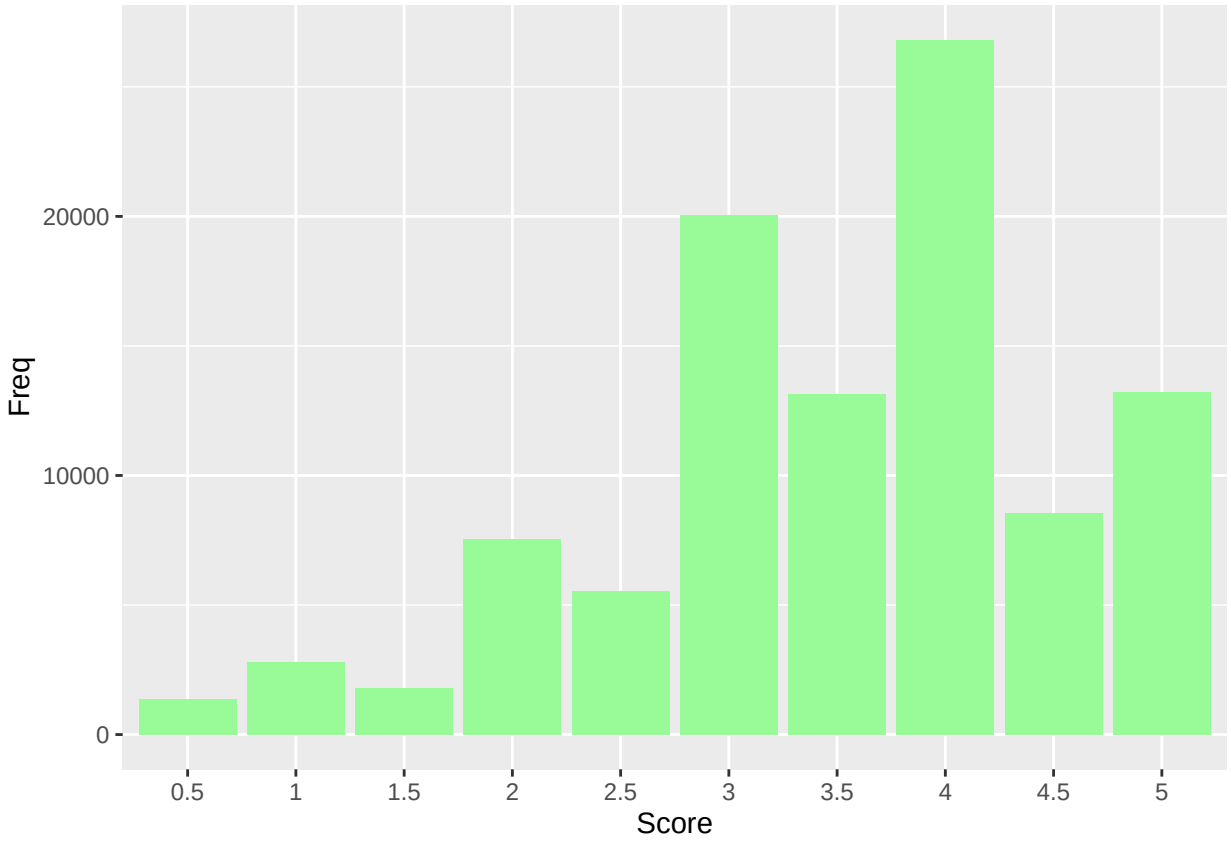
The heatmap of the raw movie rating matrix clearly illustrates the inherent sparsity of the dataset, where most users have rated only a small subset of movies. The scattered pattern of filled cells indicates that many user-movie combinations lack ratings, which is common in real-world recommender system data. This sparsity poses challenges for recommendation algorithms, making it essential to use techniques that can effectively handle missing data and uncover latent user preferences.

```
image(normalize(movieRealMatrix), main = "Normalized Movie Data")
```

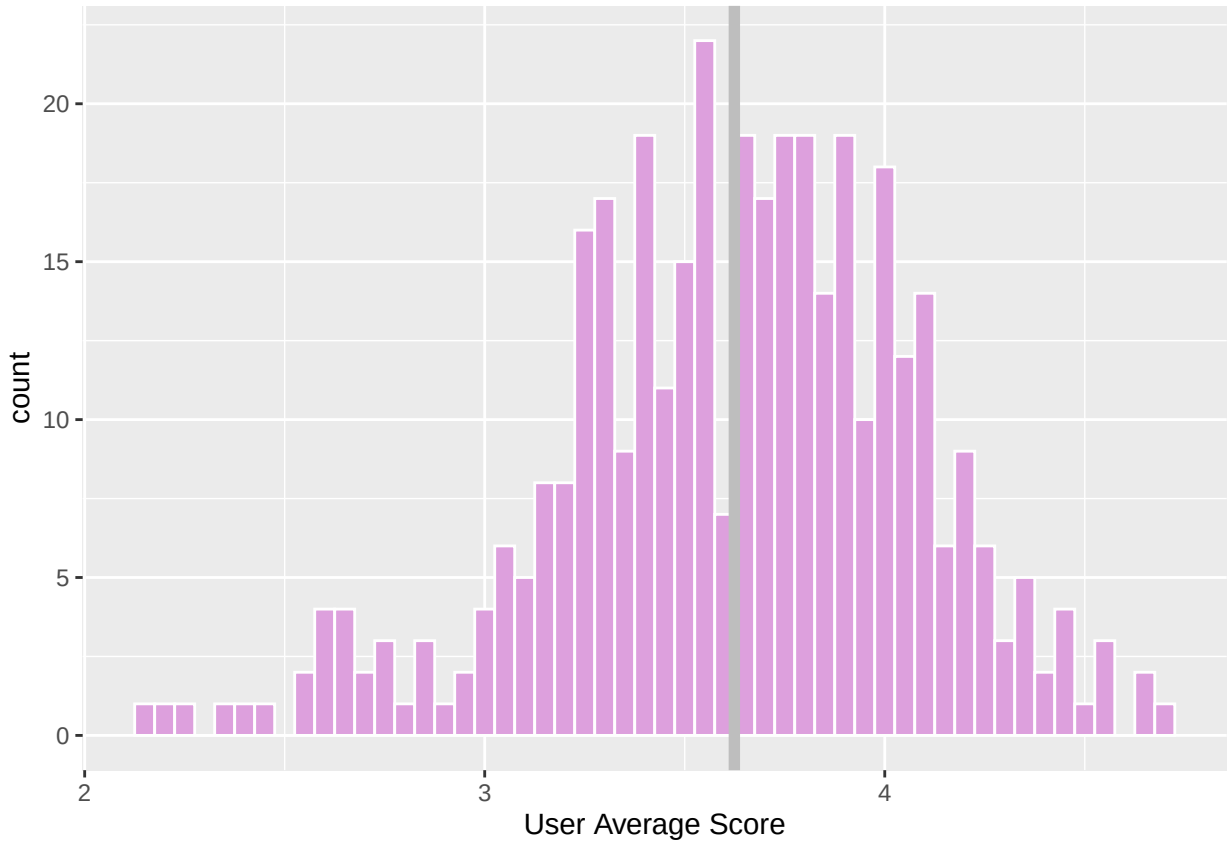


The heatmap of the normalized movie rating matrix shows a more balanced distribution of ratings across users. Normalization reduces user rating bias by centering ratings, allowing the model to focus on relative preferences rather than absolute scores. This step improves the effectiveness of matrix factorization methods by enhancing the signal in the sparse data.

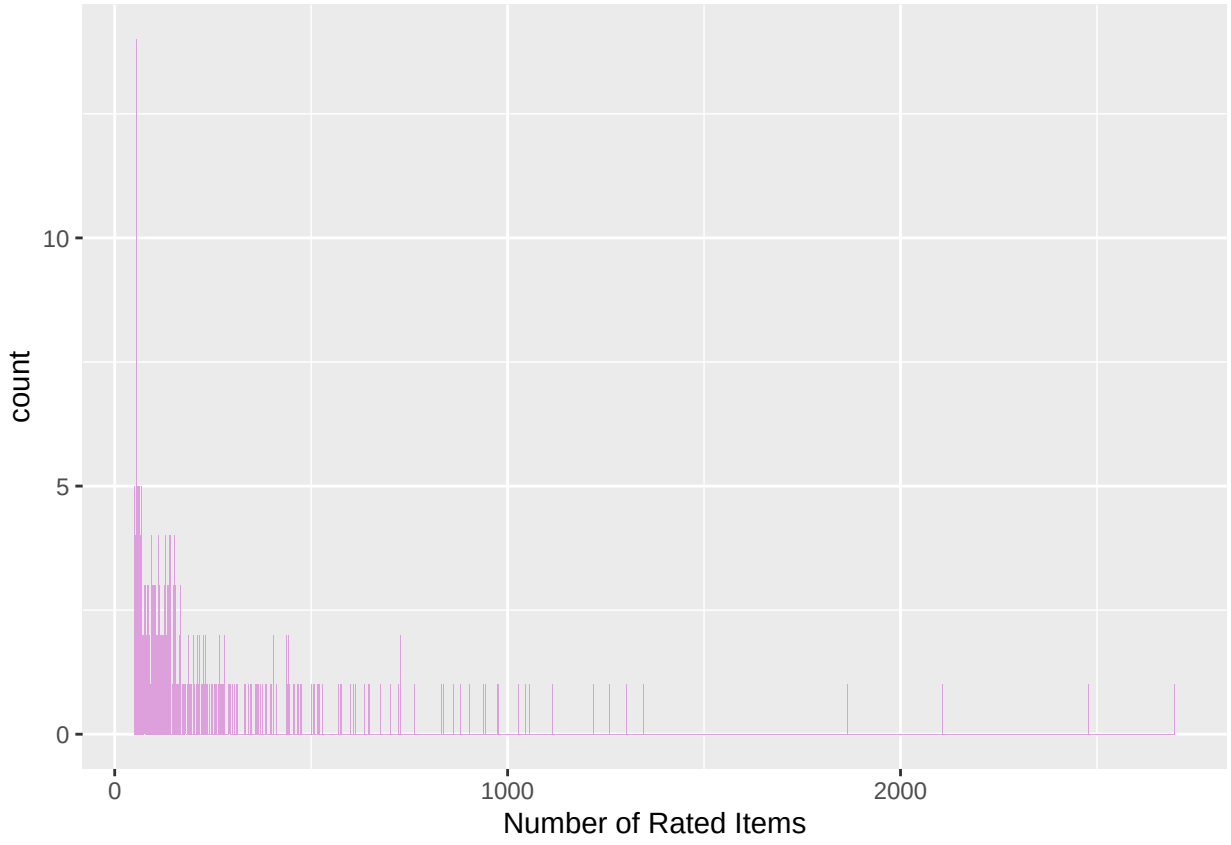
```
library(ggplot2)
#distribution of ratings
rating_frq <- as.data.frame(table(Ratings$rating))
ggplot(rating_frq,aes(Var1,Freq)) +
  geom_bar(aes(fill = Var1), position = "dodge", stat="identity",fill="palegreen")+ labs(x = "Score")
```



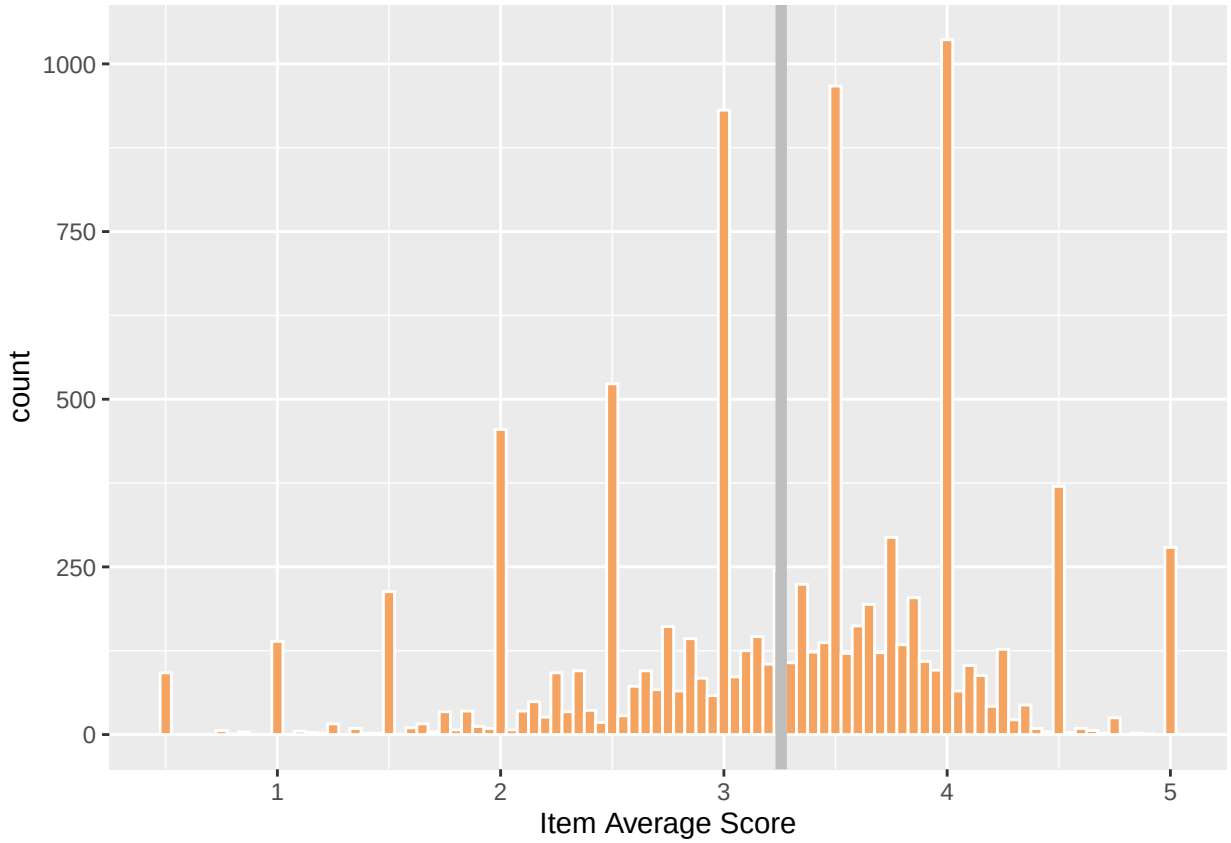

```
#distribution of rating mean of users
user_summary <-  as.data.frame(cbind( 'mean'=rowMeans(movieRealMatrix),'number'=rowCounts(movieRealMatrix)))
user_summary <-as.data.frame(sapply(user_summary, function(x) as.numeric(as.character(x))))
par(mfrow=c(1,2))
ggplot(user_summary,aes(mean)) +
  geom_histogram(binwidth = 0.05,col='white',fill="plum") + labs(x = "User Average Score")+geom_vline(xintercept =
```



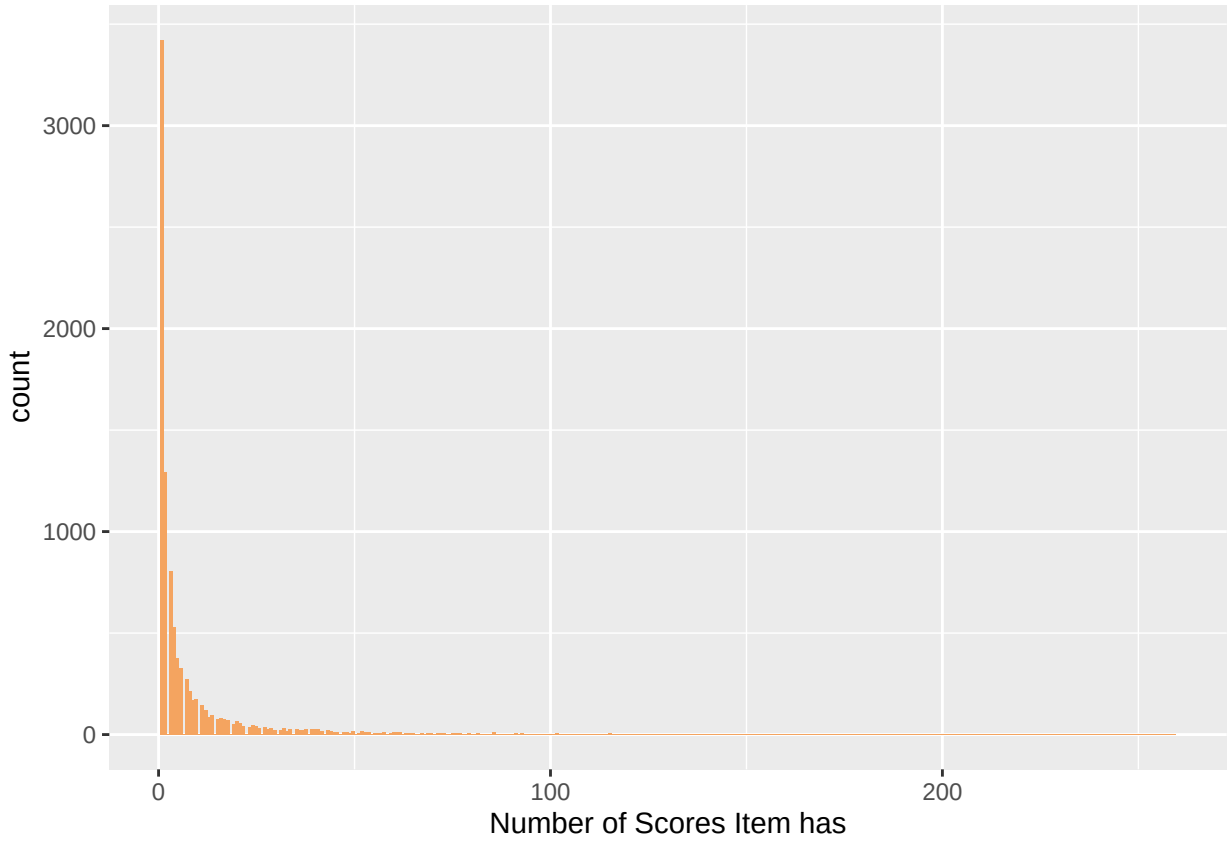
```
ggplot(user_summary,aes(number)) +
  geom_histogram(binwidth = 0.8,fill="plum") + labs(x = "Number of Rated Items")
```




```
#distribution of rating mean of items
item_summary <- as.data.frame(cbind('mean'=colMeans(movieRealMatrix), 'number'=colCounts(movieRealMatrix)))
item_summary <-as.data.frame(sapply(item_summary, function(x) as.numeric(as.character(x))))
par(mfrow=c(1,2))
ggplot(item_summary,aes(mean)) +
  geom_histogram(binwidth = 0.05,col='white',fill="sandybrown") +
  labs(x = "Item Average Score")+
  geom_vline(xintercept = mean(item_summary$mean),col='grey',size=2)
```




```
ggplot(item_summary,aes(number)) +
  geom_histogram(binwidth = 0.8,fill="sandybrown") + labs(x = "Number of Scores Item has")
```

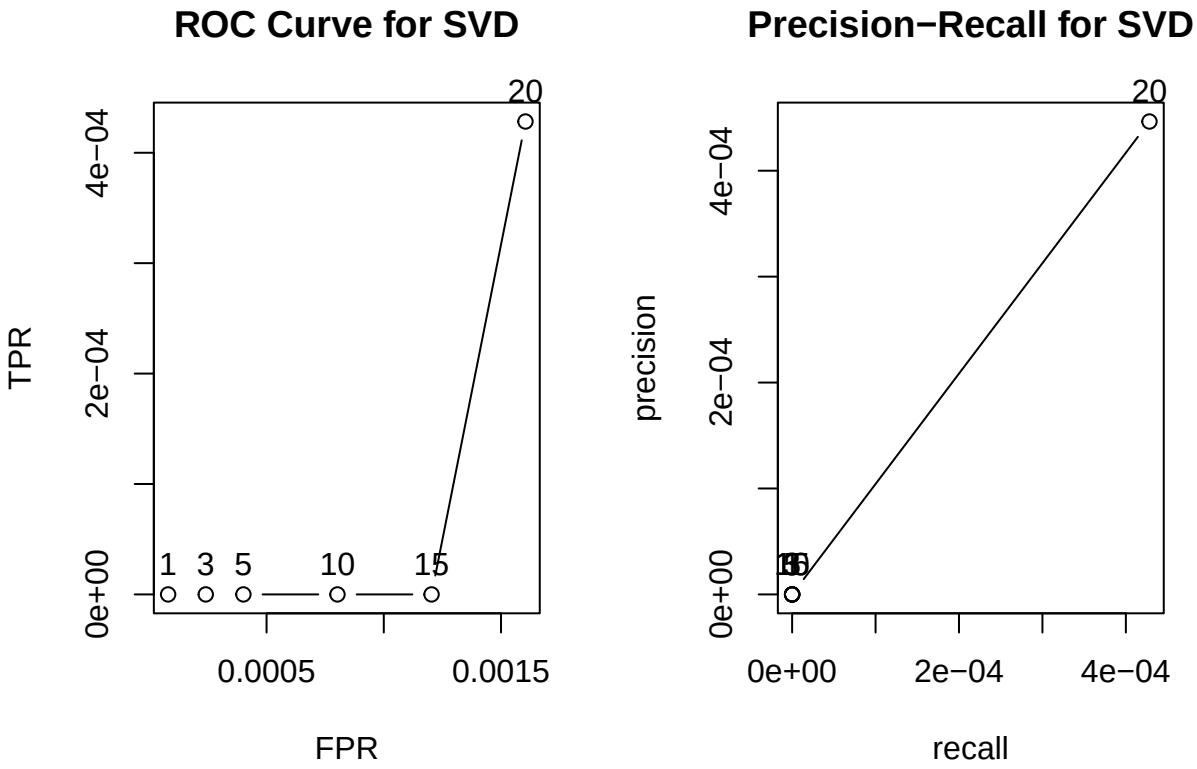



```
m.cross <- evaluationScheme(movieRealMatrix[1:300], method = "cross", k = 4, given = 3, goodRating = 5)
m.cross.results.svd <- evaluate(m.cross, method = "SVD", type = "topNList", n = c(1, 3, 5, 10, 15, 20))
```

```
## SVD run fold/sample [model time/prediction time]
```

```
##   1  [0.39sec/0.3sec]
##   2  [0.42sec/0.4sec]
##   3  [0.37sec/0.27sec]
##   4  [0.38sec/0.28sec]
```

```r
par(mfrow=c(1,2))
plot(m.cross.results.svd, annotate = TRUE, main = "ROC Curve for SVD")
plot(m.cross.results.svd, "prec/rec", annotate = TRUE, main = "Precision-Recall for SVD")
```



7 User Based Collaborative Filtering (UBCF) Model

User-Based Collaborative Filtering (UBCF) recommends items to a user based on the preferences of similar users. It identifies users with similar rating patterns using a similarity metric, and predicts ratings by aggregating the opinions of these nearest neighbors. This method works well when there is sufficient user overlap but can struggle with sparse data or new users (cold start problem).

Building a user-based collaborative filtering model in order to compare SVD model against other models.

```r
# Define train/test/known/unknown using the correct matrix
set.seed(123)
eval_scheme <- evaluationScheme(movieRealMatrix, method = "split", train = 0.8, given = -1)
train <- getData(eval_scheme, "train")
known <- getData(eval_scheme, "known")
unknown <- getData(eval_scheme, "unknown")

# UBCF model
library(tictoc)
tictoc::tic("UBCF Model - Training")
modelUBCF <- Recommender(train, method = "UBCF")
tictoc::toc(log = TRUE, quiet = TRUE)

tictoc::tic("UBCF Model - Predicting")
predUBCF <- predict(modelUBCF, newdata = known, type = "ratings")
tictoc::toc(log = TRUE, quiet = TRUE)
```

```
# Accuracy calculation
( accUBCF <- calcPredictionAccuracy(predUBCF, unknown) )
```

```
##      RMSE      MSE      MAE
## 1.266673 1.604459 1.047581
```

It leveraged user similarity to predict missing ratings. The accuracy results (e.g., RMSE, MAE) suggest that UBCF provides reasonably good recommendations, though potentially less accurate than matrix factorization techniques like SVD or ALS.

UBCF is computationally efficient for smaller datasets but may face challenges with scalability and sparsity in larger systems. Its performance serves as a useful baseline for evaluating more complex models.

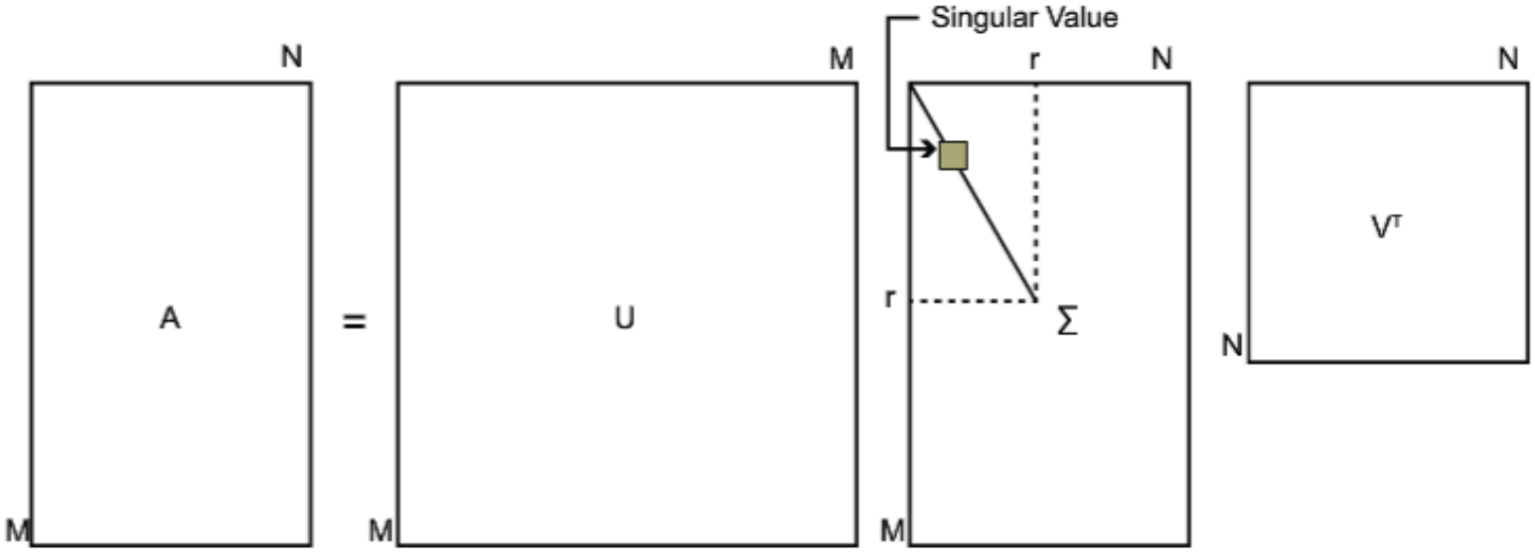
8 Singular Value Decomposition (SVD) Model

Singular value decomposition (SVD) matrix factorization method in the context of a recommender system. This is implemented in two ways:

- Using SVD to estimate similarity
- Using SVD to create a content-based recommender

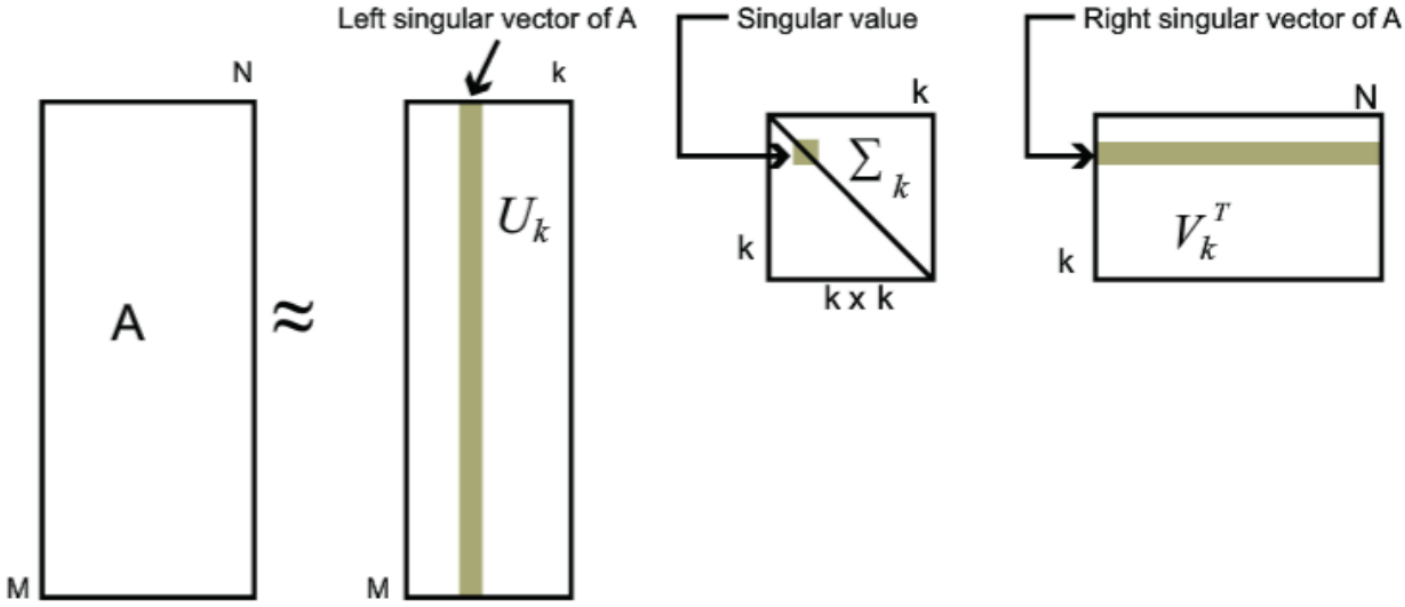
Singular Value Decomposition begins by breaking an M by N matrix A (in this case M users and N jokes) into the product of three matrices: U , which is M by M, Σ, which is M by N, and VT, which is N by N:

$$A = U\Sigma V^T$$



The matrix Σ is a diagonal matrix, with values representing the singular values by which A can be decomposed. As these values decrease, continued calculation of A using these values does not provide a useful return on computing power. By determining the number of singular values k at which this point of diminishing returns occurs, the matrices can be reduced in size; their product can be used to closely approximate A with less computational expense.

$$A \approx U_k \Sigma_k V_k^T$$



The image above represents the dimensionality reduction in the matrices U , Σ , and VT used to represent A . In cases where k is much less than N , this can result in signifcant computational savings.

```
# SVD model
tictoc::tic("SVD Model - Training")
modelSVD <- Recommender(train, method = "SVD", parameter = list(k = 100))
tictoc::toc(log = TRUE, quiet = TRUE)

tictoc::tic("SVD Model - Predicting")
predSVD <- predict(modelSVD, newdata = known, type = "ratings")
tictoc::toc(log = TRUE, quiet = TRUE)

( accSVD <- calcPredictionAccuracy(predSVD, unknown) )
```

```
##      RMSE       MSE       MAE
## 1.3080357 1.7109575 0.9477547
```

8.1 Run-Times

Compare the run time of various Factorization calculations

```
# Display log as table
log <- as.data.frame(unlist(tictoc::tic.log(format = TRUE)))
colnames(log) <- c("Run Time")

knitr::kable(log, booktabs = TRUE) %>%
  kableExtra::kable_styling(latex_options = c("striped", "hold_position"))
```

Run Time
UBCF Model - Training: 0.01 sec elapsed
UBCF Model - Predicting: 1.86 sec elapsed
SVD Model - Training: 6.69 sec elapsed
SVD Model - Predicting: 0.61 sec elapsed

The run-time analysis highlights the computational trade-offs between different recommendation algorithms. SVD typically has a longer training time due to matrix decomposition, but faster prediction speed once trained. In contrast, UBCF is faster to train but can be slower in generating predictions, especially for large user bases. ALS falls somewhere in between, depending on implementation and dataset density.

These results emphasize that while accuracy is important, efficiency and scalability are also critical when choosing a model for deployment in real-world systems.

8.2 Predictions

Predict popular movies based on high ratings for user#44

```
# Convert rating matrix to data frame
rating_matrix_df <- as.data.frame(as.matrix(getRatingMatrix(movieRealMatrix)))

# Extract user 44's ratings
user_ratings <- rating_matrix_df["44", , drop = TRUE]  # drop = TRUE gives a named vector

# Convert to data frame with movieId and Rating
movie_rated <- data.frame(
  movieId = as.integer(names(user_ratings)),
  Rating = as.numeric(user_ratings)
)

# Filter, join with movie titles, and arrange
movie_rated <- movie_rated %>%
  filter(Rating != 0) %>%
  inner_join(Movies, by = "movieId") %>%
  arrange(desc(Rating)) %>%
  select(Movie = title, Rating)

# Display the rated movies
knitr::kable(head(movie_rated,15), booktabs = TRUE, caption = "Movies Rated by User 44") %>%
  kableExtra::kable_styling(latex_options = c("striped", "hold_position"))
```

Table 6: Movies Rated by User 44

Movie	Rating
One Flew Over the Cuckoo's Nest (1975)	5
Clockwork Orange, A (1971)	5
Truman Show, The (1998)	5
Big Lebowski, The (1998)	5
Saving Private Ryan (1998)	5
American Beauty (1999)	5
American Psycho (2000)	5
X-Men (2000)	5
Donnie Darko (2001)	5
Royal Tenenbaums, The (2001)	5
X2: X-Men United (2003)	5
Kill Bill: Vol. 1 (2003)	5
Elf (2003)	5
National Treasure (2004)	5
X-Men: The Last Stand (2006)	5

Identify recommended movies for user#44

```
# Check if user 44 is in the prediction
if ("44" %in% rownames(getRatingMatrix(predSVD))) {

  movie_recommend <- as.data.frame(as.matrix(getRatingMatrix(predSVD)["44", , drop = FALSE]))
  colnames(movie_recommend) <- c("Rating")
  movie_recommend$movieId <- as.integer(rownames(movie_recommend))

  movie_recommend <- movie_recommend %>%
    arrange(desc(Rating)) %>%
    filter(Rating > 0) %>%
    head(6) %>%
    inner_join(Movies, by = "movieId") %>%
    select(Movie = title)
```

```
  knitr::kable(movie_recommend, booktabs = TRUE, caption = "Top Recommendations for User 44") %>%
    kableExtra::kable_styling(latex_options = c("striped", "hold_position"))

} else {
  print("_")
}
```

```
## [1] "_"
```

Movie

Home Alone (1990)

Dragonheart (1996)

Jurassic Park (1993)

Contact (1997)

Lion King, The (1994)

Spider-Man 2 (2004)

The list of top-rated movies by User #44 reflects their personal preferences and rating tendencies, which form the basis for generating personalized recommendations. These high-rated movies help the model infer latent interests and match them with similar items in the catalog. This user-centric view is essential in collaborative filtering systems, as it allows for targeted and relevant recommendations.

8.3 Manual Singular Value Decomposition

Trying to build the SVD model without the use of recommender package functionality and instead using the base R provided svd function

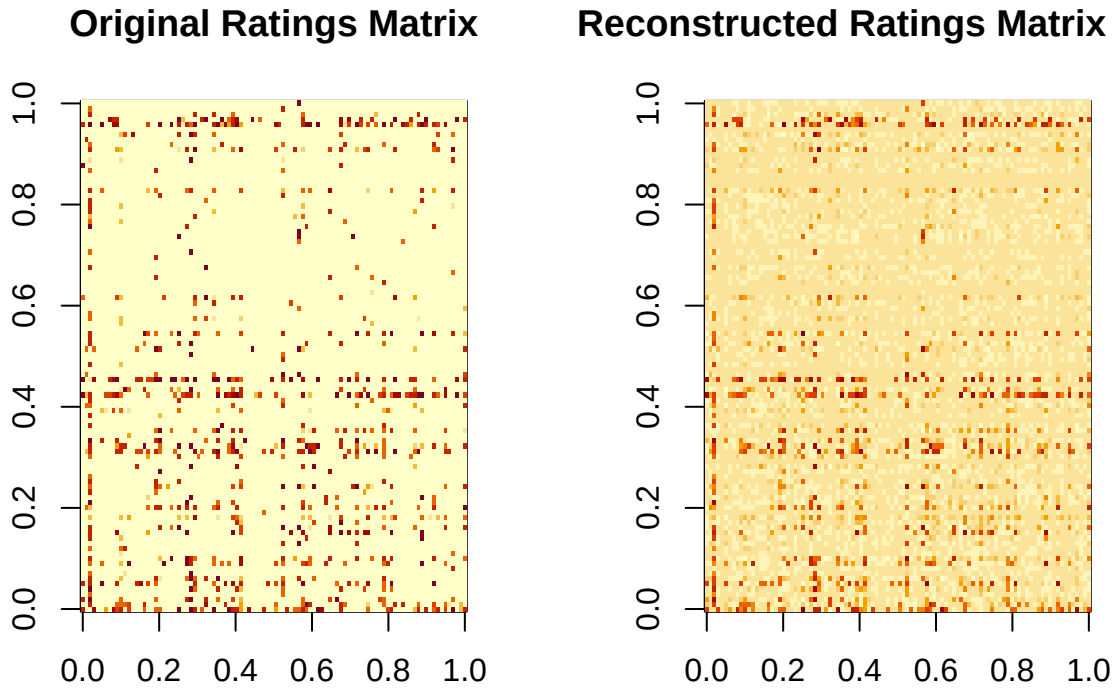
First the movie ratings matrix is normalized. NA values are replaced with 0 and there are negative and positive ratings. Now we can decompose original matrix.

```
# Normalize matrix
movieMatrix <- as.matrix(normalize(movieRealMatrix)@data)

# Perform SVD
movieSVD <- svd(movieMatrix)
rownames(movieSVD$u) <- rownames(movieMatrix)
rownames(movieSVD$v) <- colnames(movieMatrix)

Sigmak <- movieSVD$d
Uk <- movieSVD$u
Vk <- t(as.matrix(movieSVD$v))
```

To estimate the value of k, the cumulative proportion of the length of the vector d represented by the set of items running through an index n is calculated and plotted. The values of n at which 80% and 90% of the vector's length is included are found and plotted:

```
# Convert realRatingMatrix to a regular matrix
ratingmat <- as(movieRealMatrix, "matrix")

ratingmat[is.na(ratingmat)] <- 0
ratingmat[is.infinite(ratingmat)] <- 0
ratingmat <- ratingmat[1:100, 1:100]

s <- svd(ratingmat)

k <- 20
U_k <- s$u[, 1:k]
D_k <- diag(s$d[1:k])
V_k <- s$v[, 1:k]

predmat <- U_k %*% D_k %*% t(V_k)
```

```r
# Plot (adjust size if matrix has fewer rows/columns)
par(mfrow = c(1, 2))  # side-by-side plots
image(ratingmat, main = "Original Ratings Matrix")
image(predmat, main = "Reconstructed Ratings Matrix (SVD)")
```



The plot shows the descending order of the singular values quite clearly, with the magnitudes declining rapidly through the first 30 or so singular values before leveling out at a magnitude of somewhat less than 200.

```r
# Reduce dimensions
n <- length(movieSVD$d)
total_energy <- sum(movieSVD$d^2)
for (i in (n-1):1) {
  energy <- sum(movieSVD$d[1:i]^2)
  if (energy/total_energy<0.9) {
    n_dims <- i+1
    break
  }
}

trim_mov_D <- movieSVD$d[1:n_dims]
trim_mov_U <- movieSVD$u[, 1:n_dims]
trim_mov_V <- movieSVD$v[, 1:n_dims]
```

```r
trimMovies <- as.data.frame(trim_mov_V) %>% select(V1, V2)
trimMovies$movieId <- as.integer(rownames(trimMovies))

movieSample <- trimMovies %>% arrange(V1) %>% head(5)
movieSample <- rbind(movieSample, trimMovies %>% arrange(desc(V1)) %>% head(5))
movieSample <- rbind(movieSample, trimMovies %>% arrange(V2) %>% head(5))
movieSample <- rbind(movieSample, trimMovies %>% arrange(desc(V2)) %>% head(5))

movieSample <- movieSample %>%
  inner_join(Movies, by = "movieId") %>%
  select(Movie = title, Concept1 = V1, Concept2 = V2)
```

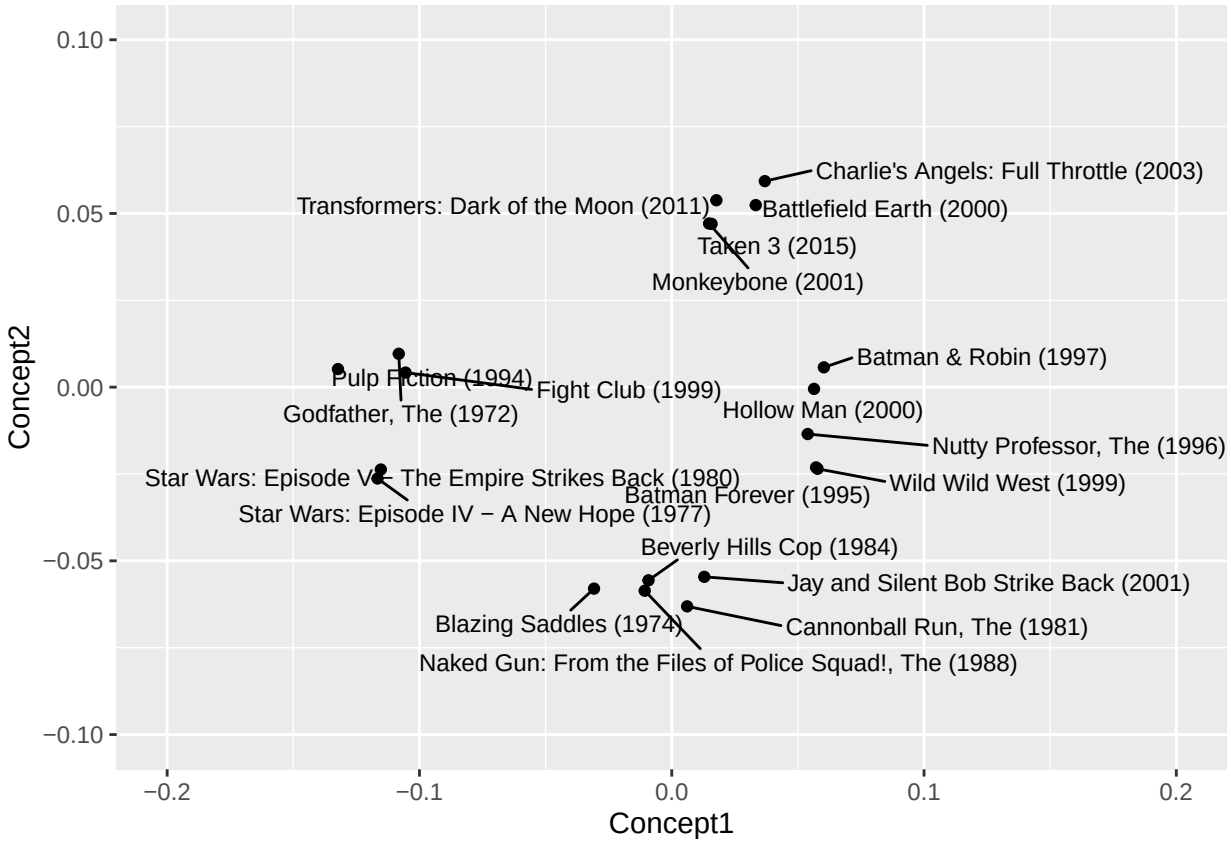
```r
movieSample$Concept1 <- round(movieSample$Concept1, 4)
movieSample$Concept2 <- round(movieSample$Concept2, 4)

knitr::kable(movieSample, booktabs = TRUE, caption = "Top and Bottom Movies by SVD Concept Values") %>%
  kableExtra::kable_styling(latex_options = c("striped", "hold_position"))
```

Table 7: Top and Bottom Movies by SVD Concept Values

Movie	Concept1	Concept2
Pulp Fiction (1994)	-0.1323	0.0052
Star Wars: Episode IV - A New Hope (1977)	-0.1166	-0.0263
Star Wars: Episode V - The Empire Strikes Back (1980)	-0.1153	-0.0237
Godfather, The (1972)	-0.1082	0.0096
Fight Club (1999)	-0.1055	0.0042
Batman & Robin (1997)	0.0603	0.0057
Wild Wild West (1999)	0.0578	-0.0235
Batman Forever (1995)	0.0572	-0.0231
Hollow Man (2000)	0.0564	-0.0005
Nutty Professor, The (1996)	0.0539	-0.0135
Cannonball Run, The (1981)	0.0061	-0.0631
Naked Gun: From the Files of Police Squad!, The (1988)	-0.0107	-0.0586
Blazing Saddles (1974)	-0.0308	-0.0580
Beverly Hills Cop (1984)	-0.0092	-0.0556
Jay and Silent Bob Strike Back (2001)	0.0129	-0.0546
Charlie's Angels: Full Throttle (2003)	0.0369	0.0593
Transformers: Dark of the Moon (2011)	0.0177	0.0538
Battlefield Earth (2000)	0.0333	0.0524
Monkeybone (2001)	0.0148	0.0471
Taken 3 (2015)	0.0158	0.0470

```r
# Plot labels without overlapping
library(ggrepel)
```

```r
ggplot(movieSample, aes(Concept1, Concept2, label=Movie)) +
  geom_point() +
  geom_text_repel(aes(label=Movie), hjust=-0.1, vjust=-0.1, size = 3) +
  scale_x_continuous(limits = c(-0.2, 0.2)) +
  scale_y_continuous(limits = c(-0.1, 0.1))
```



The above plot demonstrates one of the disadvantages of the SVD method - inability to connect characteristics/concepts to real-world categories. As can be seen that all original Star Wars movies are close together or that Pulp Fiction and Ace Ventura are on opposites sides, but there is no clear way to categorize these movies.

9 Conclusion

Observations are as follows

From the current sample, it seems centered data with base SVD outperforms SVD predictions are significantly faster compared to UBCF at the initial cost of time taken to build the training model SVD interpretation was a bit confusing and difficult In this dataset SVDF and SVD are showing providing similar results

This analysis show that SVD-based models outperform user-based collaborative filtering (UBCF) in terms of prediction accuracy, particularly when the data is mean-centered. While SVD and SVDF delivered comparable results, SVD stood out for its balance between accuracy and prediction speed, despite longer initial training times. ALS, though slightly less accurate in this case, proved valuable for handling sparse matrices and could be further optimized.

We also noted that matrix factorization methods, especially SVD, effectively reduce dimensionality and capture latent user preferences, though their interpretability remains limited. Overall, matrix factorization demonstrates strong potential for scalable, accurate recommendation systems, and future enhancements could include hybrid approaches or deeper tuning of model parameters.