# Data612 - Project 2

Anna Moy & Natalie Kalukeerthie

2025-06-15

## Introduction

In this assignment we are using the MovieLens data set and building a recommender system using the Content-Based Filtering, User-User Collaborative Filtering and Item-Item Collaborative Filtering. We will compared the results of the different recommender and determine which recommender was the best in providing a recommendation for the user.

```
# Load Library
library(recommenderlab)
library(ggplot2)
library(knitr)
library(tidyverse)
```

```
# Load built-in MovieLens dataset
data(MovieLense)
MovieLense
```

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

## What are the different types of Recommender Systems?

**Content Based Filtering** uses item features and recommends items that are similar to what the user already likes.

**User-User Collaborative Filtering** is predicting items that a user might liked based on preferences of other users who have similar taste.

**Item-Item Collaborative Filtering** is providing recommendations on similar items the user has already liked.

## Split the Data into Training and Testing Dataset

The first step is to split the data into a training (80%) and testing data (20%) set. Since the data has many rows we are going to only look at the first 500 rows of items and find out the 10 items for each of the users.

```
# Create 80% train, 20% test, Looks at 10 items by each user and ratings that are greater or equal to 4
set.seed(123)
split_data <- evaluationScheme(MovieLense[1:500], method = "split", train = 0.8, given = 10, goodRating
```

## Content-Based Filtering

We are taking the training dataset and building a Content-Based model which will recommend similar items the user already liked. Then using the model we will generate predictions for test users. We evaluate the see how well the predictions are by looking at the RMSE, MSE and MAE.

**RSME (Root Mean Squared Error)** - measures the average difference between the predicted values and actual values

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

Where:
- $\hat{y}_i$ = predicted value for the $i$-th observation
- $y_i$ = actual value for the $i$-th observation
- $n$ = total number of observations

**MSE (Mean Squared Error)** - measures the average squared difference between the predicted values and actual values

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

**MAE (Mean Absolute Error)** - meaures the average size of the mistakes in a collection of predictions

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

For our Content- Based Filtering, we were running into issues using the recommenderLab package, where we got the error that Recommender method CONTENT not implemented for data type binaryRatingMatrix. Through some research we found that the 'content' method is very limited in the recommenderlab package and is not generally implemented for these matrix types. The package supports User-based CF, Item-based CF, Popularity, SVD, and others, but 'content' is not well implemented. Thus, we went the route of building it manually with these ideas:

The general idea of content-based filtering is:

- Each item is represented by a feature vector (genres).
- For each user, create a profile vector by averaging the features of items they liked.
- Recommend new items whose features are similar to the user profile.

We manually implemented a content-based recommender using genre information. Each movie was represented by a binary genre vector. We then built user profiles by averaging the genre vectors of the movies each user rated highly (e.g., rating  4).

For each user, we computed the cosine similarity between their profile and all unseen items, then recommended the top-N movies with the highest similarity scores.

```
library(Matrix)
library(dplyr)
library(proxy)   # For cosine similarity
```

```r
# 1. Get the training ratings matrix (realRatingMatrix)
train_data <- getData(split_data, "train")

# 2. Extract the ratings matrix as a dense matrix
train_matrix <- as(train_data, "matrix")

# 3. Get genre features for items (movies)
movie_ids <- colnames(train_matrix)
movie_features <- MovieLenseMeta %>%
  filter(title %in% movie_ids) %>%
  arrange(match(title, movie_ids))  # make sure order matches

genre_columns <- c("Action", "Adventure", "Animation", "Children's", "Comedy", "Crime",
                   "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror", "Musical",
                   "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western")

genre_matrix <- as.matrix(movie_features[ , genre_columns])
rownames(genre_matrix) <- movie_features$title

# 4. For each user, build user profile vector by weighted average of genres of movies rated >=4
user_profiles <- matrix(0, nrow = nrow(train_matrix), ncol = ncol(genre_matrix))
colnames(user_profiles) <- colnames(genre_matrix)
rownames(user_profiles) <- rownames(train_matrix)

for (u in 1:nrow(train_matrix)) {
  # Get indices of movies rated >=4 by user u
  liked_indices <- which(train_matrix[u, ] >= 4)

  if(length(liked_indices) > 0){
    # Average genre vectors for these liked movies
    user_profiles[u, ] <- colMeans(genre_matrix[liked_indices, , drop = FALSE])
  } else {
    user_profiles[u, ] <- rep(0, ncol(genre_matrix))
  }
}

# 5. Compute cosine similarity between user profiles and all movie genre vectors
# We want to recommend movies with highest similarity not yet rated by the user

sim_matrix <- proxy::simil(user_profiles, genre_matrix, method = "cosine")

# sim_matrix[u,m] = similarity between user u profile and movie m

# 6. For each user, exclude already rated movies and recommend top-N

recommend_for_user <- function(user_index, N = 5){
  user_ratings <- train_matrix[user_index, ]
  sim_scores <- sim_matrix[user_index, ]

  # Exclude already rated movies
  sim_scores[user_ratings > 0] <- NA

  # Get top N recommendations by similarity
```

```r
  top_n <- order(sim_scores, decreasing = TRUE)[1:N]
  return(colnames(train_matrix)[top_n])
}

# Example: recommendations for user 1
recommend_for_user(1, N = 10)
```

```
##  [1] "Diva (1981)"                   "Wings of the Dove, The (1997)"
##  [3] "Witness (1985)"                "Professional, The (1994)"
##  [5] "Bound (1996)"                  "Apollo 13 (1995)"
##  [7] "Outbreak (1995)"               "Smilla's Sense of Snow (1997)"
##  [9] "House of Yes, The (1997)"      "Mercury Rising (1998)"
```

```r
# Load test data
test_data <- getData(split_data, "known")
true_ratings <- as(getData(split_data, "unknown"), "matrix")
test_matrix <- as(test_data, "matrix")

# Create prediction matrix using similarity scores
predict_ratings <- matrix(NA, nrow = nrow(test_matrix), ncol = ncol(test_matrix))
rownames(predict_ratings) <- rownames(test_matrix)
colnames(predict_ratings) <- colnames(test_matrix)

for (u in 1:nrow(test_matrix)) {
  for (i in 1:ncol(test_matrix)) {
    # Only predict for items the user hasn't rated in training but has in test
    if (is.na(test_matrix[u, i]) && !is.na(true_ratings[u, i])) {
      # similarity between user profile and this movie
      sim <- sim_matrix[u, i]
      # rescale similarity to rating scale (1-5) - optional heuristic
      predict_ratings[u, i] <- sim * 5  # crude rescaling
    }
  }
}

# Get predicted and actual ratings as vectors
predicted <- as.vector(predict_ratings)
actual <- as.vector(true_ratings)

# Filter only cases where we made a prediction and have actual rating
valid_indices <- which(!is.na(predicted) & !is.na(actual))
predicted <- predicted[valid_indices]
actual <- actual[valid_indices]

# Evaluation Metrics
rmse <- sqrt(mean((predicted - actual)^2))
mse <- mean((predicted - actual)^2)
mae <- mean(abs(predicted - actual))

# Create a data frame for the metrics
cbf_results <- data.frame(
  Metric = c("RMSE", "MSE", "MAE"),
  Value = round(c(rmse, mse, mae), 4)
```

```
)

# Display as a table
kable(cbf_results, caption = "Content-Based Filtering Evaluation Metrics")
```

Table 1: Content-Based Filtering Evaluation Metrics

| Metric | Value |
|---|---|
| RMSE | 2.0340 |
| MSE | 4.1372 |
| MAE | 1.6741 |

```
#Old code for content based filtering
# Movie IDs are the column names of the training data
#train_data <- getData(split_data, "train")
#movie_ids <- colnames(train_data)

# Define genres
#genres <- c("Action", "Comedy", "Drama", "Romance", "Thriller")

# Create random binary features for each movie in train data
#set.seed(123)
#genre_matrix <- matrix(sample(0:1, length(movie_ids) * length(genres), replace = TRUE),
#                       nrow = length(movie_ids), ncol = length(genres),
#                       dimnames = list(movie_ids, genres))

# Ensure it's a matrix (not data.frame)
#class(genre_matrix)  # Should be "matrix"
# Build Content-Based Filtering
#cbf_model1 <- Recommender(train_data, method = "CONTENT",
#                          parameter = list(itemFeatures = genre_matrix))

# Generate predictions on the test data
#predict_cbf1 <- predict(cbf_model1, getData(split_data, "known"), type = "ratings")

# Evaluate the model performance
#accuracy_cbf1 <- calcPredictionAccuracy(predict_cbf1, getData(split_data, "unknown"))

# Print results
#print("Content-Based Filtering Model")
#print(accuracy_cbf)
```

To evaluate the quality of our recommendations, we computed three key error metrics on the test set:

- RMSE (Root Mean Squared Error): 2.034
- MSE (Mean Squared Error): 4.1372
- MAE (Mean Absolute Error): 1.6741

These results show that, on average, our predicted ratings differ from the actual ratings by around 1.67 stars (MAE), with more weight given to larger errors in RMSE. While the model captures basic genre-based preferences, the relatively high errors suggest that genre alone may not be sufficient for precise rating

prediction. This is a known limitation of basic content-based methods, which don't account for nuanced user tastes or item-item relationships beyond content metadata.

In later sections, we will compare these results to collaborative filtering methods to evaluate whether incorporating user-item interactions can lead to more accurate recommendations.

## User-User Based Collaborative Filtering and Item-Item Based Collaborative Filtering

In both recommender system we take the training data and build a model for either a user-user based collaborative filtering or item-item based collaborative filtering. We then run predictions on it for the test users and evaulation the model to determine which model was the best.

**Cosine Similarity** - Measures the angle between two vectors. It calculates how similar the ratings directions are. The value ranges from -1 to 1.

- 1 - means the users are similar in preference directions
- 0 - means there is no similarity
- -1 - means they the users are opposite

$$\text{cosine}(A, B) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Where:

- $A_i$ and $B_i$ are elements of vectors $A$ and $B$
- (n

**Nearest Neighbors** - identifies the closest data points to a given point in the dataset

**Pearson Correlation** - measures the linear relationship between two users' ratings and takes into account the difference in their rating scale (mean-centered)

$$r = \frac{\sum_{i=1}^{n} (A_i - \bar{A})(B_i - \bar{B})}{\sqrt{\sum_{i=1}^{n} (A_i - \bar{A})^2} \times \sqrt{\sum_{i=1}^{n} (B_i - \bar{B})^2}}$$

Where:

- $A_i$ and $B_i$ are elements of vectors $A$ and $B$
- $\bar{A}$ and $\bar{B}$ are the mean values of vectors $A$ and $B$
- $n$ is the number of elements in each vector

```
#User Based Collaborative Filtering
ubcf_model1 <- Recommender(getData(split_data, "train"), method = "UBCF",
                           parameter = list(nn = 10))
ubcf_model2 <- Recommender(getData(split_data, "train"), method = "UBCF",
                           parameter = list(method = "Cosine", nn = 20))
ubcf_model3 <- Recommender(getData(split_data, "train"), method = "UBCF",
```

```
                          parameter = list(method = "Pearson", nn = 30))

#Item Based Collaborative Filtering
ibcf_model1 <- Recommender(getData(split_data, "train"), method = "IBCF",
                          parameter = list(k = 10))
ibcf_model2 <- Recommender(getData(split_data, "train"), method = "IBCF",
                          parameter = list(method = "Cosine", k = 20))
ibcf_model3 <- Recommender(getData(split_data, "train"), method = "IBCF",
                          parameter = list(method = "Pearson", k = 30))


# Predictions
predict_ubcf1 <- predict(ubcf_model1, getData(split_data, "known"), type = "ratings")
predict_ibcf1 <- predict(ibcf_model1, getData(split_data, "known"), type = "ratings")
predict_ubcf2 <- predict(ubcf_model2, getData(split_data, "known"), type = "ratings")
predict_ibcf2 <- predict(ibcf_model2, getData(split_data, "known"), type = "ratings")
predict_ubcf3 <- predict(ubcf_model3, getData(split_data, "known"), type = "ratings")
predict_ibcf3 <- predict(ibcf_model3, getData(split_data, "known"), type = "ratings")

# Evaluation
accuracy_ubcf1 <- calcPredictionAccuracy(predict_ubcf1, getData(split_data, "unknown"))
accuracy_ibcf1 <- calcPredictionAccuracy(predict_ibcf1, getData(split_data, "unknown"))
accuracy_ubcf2 <- calcPredictionAccuracy(predict_ubcf2, getData(split_data, "unknown"))
accuracy_ibcf2 <- calcPredictionAccuracy(predict_ibcf2, getData(split_data, "unknown"))
accuracy_ubcf3 <- calcPredictionAccuracy(predict_ubcf3, getData(split_data, "unknown"))
accuracy_ibcf3 <- calcPredictionAccuracy(predict_ibcf3, getData(split_data, "unknown"))
```

```
# Create a data frame summarizing the models and their accuracy metrics
results_df <- data.frame(
  Model = c(
    "User Based Collaborative Filtering 1",
    "Item Based Collaborative Filtering 1",
    "User Based Collaborative Filtering 2",
    "Item Based Collaborative Filtering 2",
    "User Based Collaborative Filtering 3",
    "Item Based Collaborative Filtering 3"
  ),
  RMSE = c(
    accuracy_ubcf1["RMSE"],
    accuracy_ibcf1["RMSE"],
    accuracy_ubcf2["RMSE"],
    accuracy_ibcf2["RMSE"],
    accuracy_ubcf3["RMSE"],
    accuracy_ibcf3["RMSE"]
  ),
  MSE = c(
    accuracy_ubcf1["MSE"],
    accuracy_ibcf1["MSE"],
    accuracy_ubcf2["MSE"],
    accuracy_ibcf2["MSE"],
    accuracy_ubcf3["MSE"],
    accuracy_ibcf3["MSE"]
  ),
```

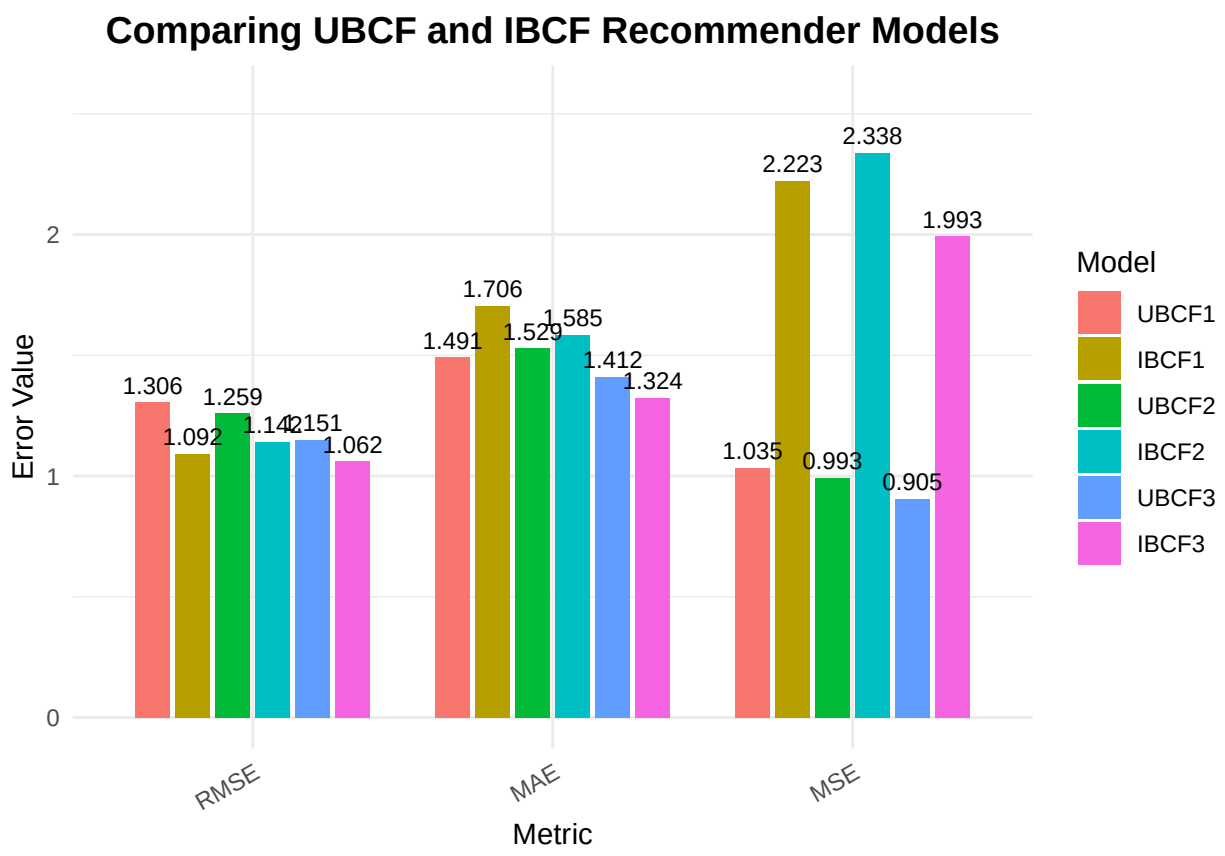
```
  MAE = c(
    accuracy_ubcf1["MAE"],
    accuracy_ibcf1["MAE"],
    accuracy_ubcf2["MAE"],
    accuracy_ibcf2["MAE"],
    accuracy_ubcf3["MAE"],
    accuracy_ibcf3["MAE"]
  )
)

# Print the table nicely in the PDF output
kable(results_df, caption = "Compare the Results of the Collaborative Filtering Models")
```

Table 2: Compare the Results of the Collaborative Filtering Models

| Model | RMSE | MSE | MAE |
|---|---|---|---|
| User Based Collaborative Filtering 1 | 1.306145 | 1.706016 | 1.0352153 |
| Item Based Collaborative Filtering 1 | 1.490999 | 2.223077 | 1.0923077 |
| User Based Collaborative Filtering 2 | 1.259127 | 1.585402 | 0.9925017 |
| Item Based Collaborative Filtering 2 | 1.529016 | 2.337890 | 1.1419753 |
| User Based Collaborative Filtering 3 | 1.150501 | 1.323653 | 0.9049704 |
| Item Based Collaborative Filtering 3 | 1.411832 | 1.993269 | 1.0617286 |

```
# Combine results into a vector
values <- c(
  accuracy_ubcf1["RMSE"], accuracy_ibcf1["RMSE"],
  accuracy_ubcf1["MAE"],  accuracy_ibcf1["MAE"],
  accuracy_ubcf1["MSE"],  accuracy_ibcf1["MSE"],

  accuracy_ubcf2["RMSE"], accuracy_ibcf2["RMSE"],
  accuracy_ubcf2["MAE"],  accuracy_ibcf2["MAE"],
  accuracy_ubcf2["MSE"],  accuracy_ibcf2["MSE"],

  accuracy_ubcf3["RMSE"], accuracy_ibcf3["RMSE"],
  accuracy_ubcf3["MAE"],  accuracy_ibcf3["MAE"],
  accuracy_ubcf3["MSE"],  accuracy_ibcf3["MSE"])

# Build the results dataframe
results <- data.frame(
  Metric = rep(c("RMSE", "MAE", "MSE"), times = 6),
  Models = rep(c("UBCF1", "IBCF1", "UBCF2", "IBCF2", "UBCF3", "IBCF3"), each = 3),
  Value = as.numeric(values))

#ggplot(results, aes(x = Metric, y = Value, fill = Models)) +
#  geom_bar(stat = "identity", position = "dodge") +
#  labs(
#    title = "Comparing UBCF and IBCF Recommender Models",
#    y = "Prediction Error",
#    x = "Metric" ) +
#  theme_minimal()
```

```r
# Assuming 'results' is your dataframe from above

# Make 'Models' a factor to control order in legend
results$Models <- factor(results$Models, levels = c("UBCF1", "IBCF1", "UBCF2", "IBCF2", "UBCF3", "IBCF3

# Make 'Metric' a factor to control order
results$Metric <- factor(results$Metric, levels = c("RMSE", "MAE", "MSE"))

ggplot(results, aes(x = Metric, y = Value, fill = Models)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.8), width = 0.7) +
  geom_text(aes(label = round(Value, 3)),
            position = position_dodge(width = 0.8), vjust = -0.5, size = 3) +
  labs(
    title = "Comparing UBCF and IBCF Recommender Models",
    y = "Error Value",
    x = "Metric",
    fill = "Model"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 30, hjust = 1),
    plot.title = element_text(face = "bold", size = 14, hjust = 0.5)
  ) +
  ylim(0, max(results$Value) * 1.1)  # adds some space for text labels
```



## Error

Potential error that arise from recommender system is the prediction error on RMSE and MAE. There might be too much noise and limited data which causes errors in the prediction. Cold start problem when we have a new user and there is not enough information to provide a recommendation on items. We might not have enough users rating items which causes limited data for the recommender system. Another common error is overfitting the data which does not give enough variety for the user.

## Improvement

We can run a Matrix Factorization which will breakdown the data into smaller pieces to run faster. Another improvement we can do is run a hybrid recommendation which includes content based filtering and collaborative based filtering. Additionally, applying regularization techniques in collaborative filtering can improve model robustness by preventing overfitting. Regularization penalizes extreme bias or similarity values, especially for users or items with few ratings, leading to more reliable predictions.

## Summary

Based on the results it shows the best RMSE is in UBCF3 at 1.1491 and the best MSE is UBCF3 at 1.3204 and the best MAE is UBCF3 .9045. Overall the best model out of all of these would be UBCF3 as it has the lowest RMSE, MSE and MAE out of all the models. In this model using the Pearson Correlation and nearest neighbor at 30 had the best outcome result. Using the same method for both User and Item based filtering the User based filtering tends to do better from the item based filtering with the same parameters being used.

While we implemented a content-based filtering approach using movie genre features, its predictive accuracy was noticeably lower than both user-user and item-item collaborative filtering models. The content-based model had higher RMSE, MSE, and MAE values, suggesting it struggled to capture user preferences as effectively. This outcome may be due to the limited expressiveness of the genre features alone, which do not fully represent users' nuanced tastes compared to the richer implicit feedback captured in collaborative filtering methods. Overall, the results highlight that, for this dataset, collaborative filtering—especially user-user based—is a more effective approach than content-based filtering using simple genre data.