Application of User-Based Collaborative Filtering for Streaming Recommendations
MS Machine Learning (MRKT B9653) Research Paper
Aryan Mishra

Introduction & Motivation

Recommendation systems are an interesting and integral part of modern machine learning systems that generally deal with rankings and ratings of various users and items. In fact, the growing popularity of the Internet as a medium for electronic and business transactions has fueled the advancement of recommender system technology. The simplicity with which the Web allows consumers to submit feedback about their likes and dislikes is a key catalyst in this regard. This is especially important for streaming platforms such as Netflix and Disney+, whose business models are based on monthly subscriptions. In this accord, recommender systems help maintain and improve user retention by ensuring that users receive regular recommendations that align with their preferences. Apart from improving user retention, recommendation systems also allow streaming platforms to easily analyze the market by discovering user preferences. By leveraging user ratings and the number of people watching a particular movie, streaming platforms can recommend similar movies, which can enhance user experience and lead to additional sales and profit for streaming platforms in general.

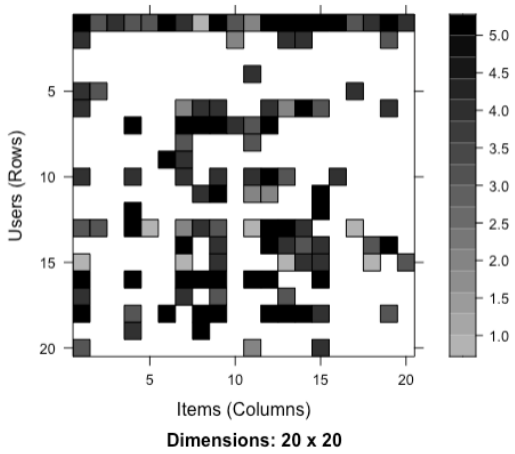
Taking these observations into account, the goal of this research paper is to explore and analyze the application of a particular type of recommendation technique, called User-Based Collaborative Filtering (UBCF), for streaming recommendations (in particular movie recommendations). Throughout this paper, we will first look at how the data was acquired and perform some basic exploratory data analysis. This will be then be followed by UBCF modeling, predictions and evaluation, conclusion, and possible improvements to the recommendation system.

Data Acquisition

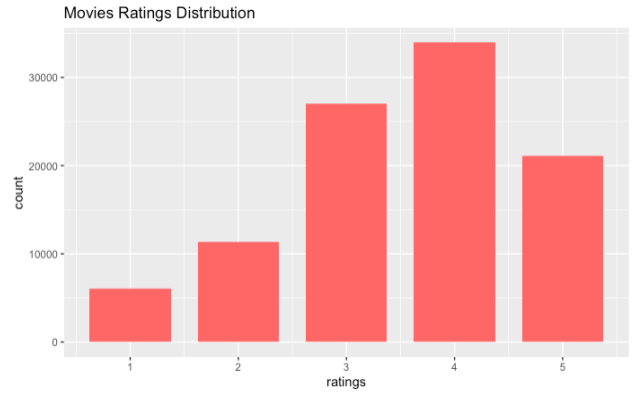
The dataset used in the project is a subset of the MovieLens dataset. The complete MovieLens dataset consists of 27 million ratings (rated on a scale from 1 to 5) of 58,000 movies by 280,000 users. Due to computational constraints, the research presented in this paper is based on a subset of this dataset with 99,392 ratings of 1,664 movies by 943 users. To acquire the data, R's *recommenderLab* package was used. This package is extremely useful in this case since it provides a research infrastructure to test and develop various recommendation algorithms, in addition to providing pre-loaded datasets for learning and exploration. Furthermore, loading the dataset from *recommenderLab* automatically creates a ratings matrix from the *realRatingMatrix* class, which is an object class created within *recommenderLab* for efficient storage of user-item ratings matrices. The *realRatingMatrix* is already optimized for storing sparse matrices, and in fact, is about nine times more efficient in conserving memory than a traditional matrix object!

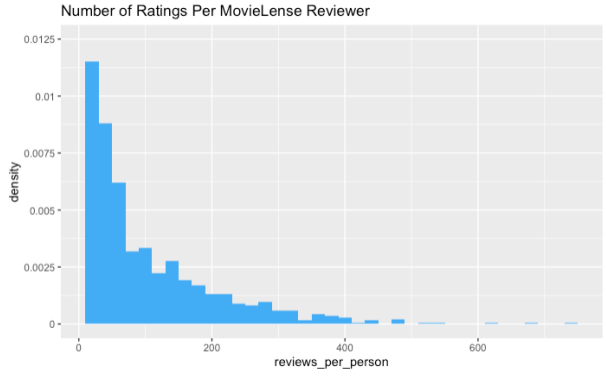
Exploratory Data Analysis

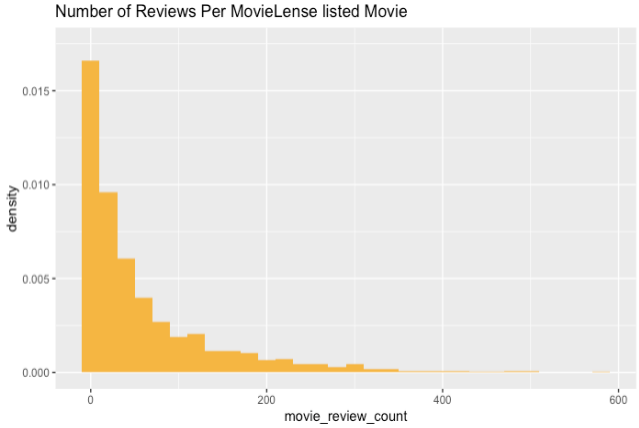
Before building the recommendation system, it is always useful to explore the dataset and understand some key trends and highlights that can help us provide actionable insights. For example, looking at the user data, we notice that most movie reviewers were men, with 670 total male ratings compared to only 273 female ratings. Interestingly, the average user age in the dataset is 34, with the youngest rater aged 7 and the oldest aged 73. When it comes to the movies, all of the rated movies were released between 1922 and 1998, with the average year being 1989 and Drama, Comedy, and Action being the most popular genres. Furthermore, we also visualized a subset of the ratings matrix to gain a preview of the user-movie ratings and visualize its sparse nature. For example, the matrix below shows the disproportionate nature of the number of ratings per user, with some users (e.g. user one) rating more movies than other users (e.g. user three).

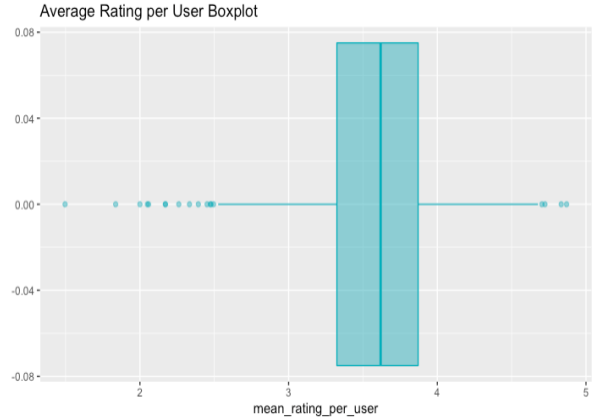


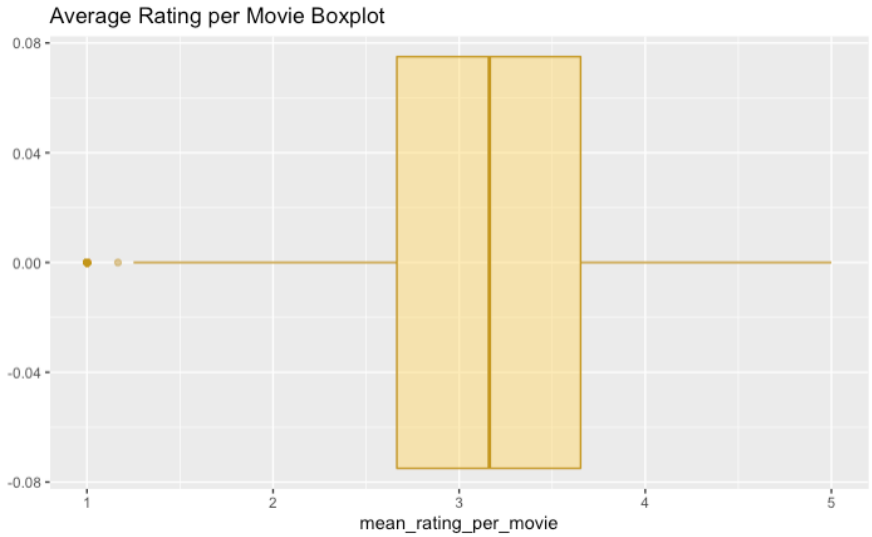
Further data visualizations were also performed, which are shown below.











Some key observations from the plots above include:

- From the general movie ratings distribution plot, we can see that the ratings are skewed to the left, with the median rating being 4 out of 5, indicating that most of the ratings were somewhat positive.
- The number of ratings per MovieLens reviewer is heavily skewed to the right, with the mean number of ratings per user being 105 and the median being 64. This is not surprising since, in almost every recommendation case study, it is rare to find users who have rated all the items, resulting in sparse ratings matrices. Note that whenever the median is smaller than the mean, the data is skewed to the right and vice versa.
- Similarly, the number of reviews per MovieLens listed movie is also heavily skewed to the right, with the mean being 59.73 and the median number being 27. This is again not surprising since we don't expect every movie to have been reviewed.
- From the boxplots, we can see that the average rating per user is centered around 3.6, and the average rating per movie is centered slightly less around 3.1. There are also more outliers present towards the left in the first boxplot, indicating that for some users, almost every movie they rated was low.

Modeling, Predictions, and Evaluation

After performing exploratory data analysis, we now proceed to the modeling phase. The focus of the modeling phase is centered around User-Based Collaborative Filtering. UBCF is an example of a neighborhood-based collaborative filtering algorithm, which are also referred to as memory-based algorithms. These were, in fact, one of the earliest algorithms developed for collaborative filtering. The idea behind collaborative filtering is that similar users tend to display similar patterns when it comes to rating behavior. UBCF is a type of neighborhood-based algorithm where ratings provided by similar users to a target user X are used to make

recommendations for X. The predicted ratings of X are computed as the weighted average of the "peer-group" ratings for each item, which in this case is a movie. For example, if John and Jane are similar users in the sense that they have rated movies in a similar fashion in the past, then we can use Jane's rating on *Toy Story* to predict John's unobserved rating on that movie. As such, the q-most similar users to John can be used to make rating predictions for John. Now that we have a high-level overview of how UBCF works, let's dive deeper into some of the parameters for this model. UBCF has four main key parameters that we need to consider:

- *Method*: This is essentially the similarity measure between the rating vector of two users u and v. In our case, we will use the *Cosine Similarity* metric, a popular metric which measures the similarity between two vectors of an inner product space. The formula is shown below:

$$Similarity(U,V) = \frac{U \cdot V}{||U|| \cdot ||V||} = \frac{\sum_{I=1}^{n} u_i \times v_i}{\sqrt{\sum_{I=1}^{n} u_i^2} \times \sqrt{\sum_{I=1}^{n} v_i^2}}$$

- *NN:* This parameter sets the neighborhood of the most similar users to consider for each target user profile.
- *Sample:* A logical value to indicate whether or not to sample the data. For this project, we didn't need this parameter since we explicitly set a reproducible seed and performed the train/test split before fitting the model.
- *Normalize:* The method to normalize the real ratings provided by different users. This is an important parameter since it accounts for user bias in movie ratings. In our case, we will *zero-mean-center* the ratings, where each user's vector of ratings is subtracted by its own mean to center the mean rating at zero.

Taking these parameters into account, the UBCF algorithm is run as follows:

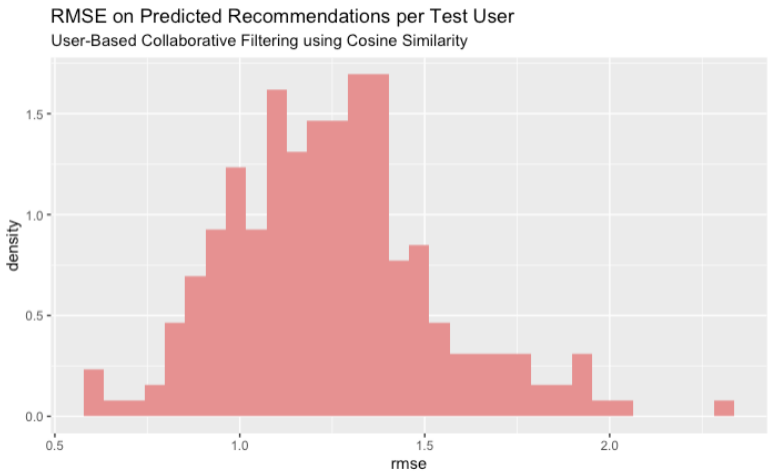
1. Compute the similarity between each user using the *Cosine Similarity* measure.
    a. For every user, identify the top q similar users. Here, q is set to 10.
2. For each movie, average the ratings by each user's 10 most similar users.
    a. Take the weighted average of the ratings based on the similarity score of each user who rated the item. The similarity score can equal the weight or we can also use any other pythagorean average (e.g. geometric, harmonic). In our case, we will just use the weight.
3. Select a top-N recommendation threshold. .

After thoroughly understanding how the model works along with the parameters, the next step was to set up the modeling training and testing scheme. In our case, we performed a random train/test split and we fitted the recommender on 75% of the data (training set) and tested the model on the remaining 35% of unseen data (test set). After this step, we are able to see the top 10

movies recommender per test user. For example, the table below shows the top 10 movies recommended to the first test user.

```
 [1] "Mrs. Brown (Her Majesty, Mrs. Brown) (1997)" "Singin' in the Rain (1952)"                "Dead Man Walking (1995)"
 [4] "Ridicule (1996)"                             "Ulee's Gold (1997)"                      "Wings of the Dove, The (1997)"
 [7] "Michael Collins (1996)"                      "Titanic (1997)"                          "Rear Window (1954)"
[10] "Twelve Monkeys (1995)"
```

Finally, we evaluated the model performance using some well-known metrics such as the RMSE, MSE, and MAE. The table below show the overall RMSE, MSE, and MAE of our model, and the plot shows the distribution of the RMSE on predicted recommendations per test user.




```
     RMSE       MSE       MAE
1.2639355 1.5975330 0.9930037
```

As we can see, the overall RMSE, MSE, and MAE of our recommendation system are 1.264, 1.598, and 0.993 respectively. However, it is hard to gauge the effectiveness of a recommender system without having another model to compare the scores with. For comparison's sake, let's use Netflix. When Netflix launched an open competition for the best collaborative filtering algorithm to predict user ratings for films in 2006, their own *CineMatch* algorithm scored an RMSE of 0.9525 on their test set, while the winning team in 2009 (*"BellKor's Pragmatic Chaos"*) scored an RMSE of 0.8597. Compared to both scores, our basic UBCF model performs considerably worse, which is unsurprising since we didn't fully optimize the model and only applied it to a small subset of the *MovieLens* data.

Conclusion & Possible Improvements

In conclusion, throughout this research paper, we explored and analyzed how UBCF works and applied it to a subset of the *MovieLens* dataset. In addition, we managed to find the top 10 movie recommendations for each user in the test set and obtained RMSE, MSE, and MAE values of 1.264, 1.598, and 0.993 respectively. There is a multitude of ways we could have improved our model. For example, we could have performed k-fold cross-validation to potentially improve model performance and use other more relevant evaluation techniques such as hit rate. Furthermore, collaborative filtering techniques don't really work well if data sparsity is too high, so we could have also experimented with Content-Based Filtering methods and other hybrid approaches such as Content-Boosted Collaborative Filtering (CBCF).

References

Aggarwal, Charu C. *Recommender Systems: The Textbook*. Springer, 2018.

Hoeft, Brandon, Recommender-System-R-Tutorial, 2017, RecommenderLab_Tutorial https://github.com/BrandonHoeft/Recommender-System-R-Tutorial/blob/master/RecommenderLab_Tutorial.Rmd

Appendix

```
#Loading the relevant libraries

library(dplyr)

library(ggplot2)

library(recommenderlab)

setwd('/Users/aryan/Desktop/MSBA_Spring_22/MRKT_9653_ML/Research_Paper')

data(MovieLense) # loads dataset

class(MovieLense)

## [1] "realRatingMatrix"

## attr(,"package")

## [1] "recommenderlab"

movie_ratings <- MovieLense

remove(MovieLense)

library(pryr)

object_size(movie_ratings) #Calculating size of realRatingMatrix object. Almost 9 times more efficient!

## 1,409,320 B

object_size(as(movie_ratings, "matrix"))

## 12,761,360 B

head(MovieLenseMeta)

##                                 title year

## 1                    Toy Story (1995) 1995
```

```
## 2                                GoldenEye (1995) 1995
## 3                            Four Rooms (1995) 1995
## 4                            Get Shorty (1995) 1995
## 5                                Copycat (1995) 1995
## 6 Shanghai Triad (Yao a yao yao dao waipo qiao) (1995) 1995
##                                                 url unknown Action
## 1     http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)       0      0
## 2     http://us.imdb.com/M/title-exact?GoldenEye%20(1995)        0      1
## 3     http://us.imdb.com/M/title-exact?Four%20Rooms%20(1995)     0      0
## 4     http://us.imdb.com/M/title-exact?Get%20Shorty%20(1995)     0      1
## 5          http://us.imdb.com/M/title-exact?Copycat%20(1995)     0      0
## 6 http://us.imdb.com/Title?Yao+a+yao+yao+dao+waipo+qiao+(1995)       0      0
##   Adventure Animation Children's Comedy Crime Documentary Drama Fantasy
## 1     0         1         1         1         0         0         0         0
## 2     1         0         0         0         0         0         0         0
## 3     0         0         0         0         0         0         0         0
## 4     0         0         0         1         0         0         1         0
## 5     0         0         0         0         1         0         1         0
## 6     0         0         0         0         0         0         1         0
##   Film-Noir Horror Musical Mystery Romance Sci-Fi Thriller War Western
## 1     0         0         0         0         0         0         0  0     0
## 2     0         0         0         0         0         0         1  0     0
## 3     0         0         0         0         0         0         1  0     0
## 4     0         0         0         0         0         0         0  0     0
## 5     0         0         0         0         0         0         1  0     0
## 6     0         0         0         0         0         0         0  0     0
```

```
head(MovieLenseUser)
##   id age sex occupation zipcode
## 1  1  24   M technician   85711
## 2  2  53   F    other   94043
## 3  3  23   M     writer   32067
## 4  4  24   M technician   43537
## 5  5  33   F    other   15213
## 6  6  42   M  executive   98101
summary(MovieLenseUser)
##        id         age       sex                occupation        zipcode
##  Min.   :  1.0   Min.   : 7.00   F:273   student    :196   55414  :  9
##  1st Qu.:236.5   1st Qu.:25.00   M:670   other      :105   55105  :  6
##  Median :472.0   Median :31.00           educator           : 95   10003  :  5
##  Mean   :472.0   Mean   :34.05           administrator: 79   20009  :  5
##  3rd Qu.:707.5   3rd Qu.:43.00           engineer           : 67   55337  :  5
##  Max.   :943.0   Max.   :73.00           programmer   : 66   27514  :  4
##                                  (Other)            :335   (Other):909
summary(MovieLenseMeta)
##        title                year      url                 unknown
##  Length:1664           Min.   :1922   Length:1664        Min.   :0.000000
##  Class :character   1st Qu.:1992   Class :character   1st Qu.:0.000000
##  Mode  :character   Median :1995   Mode  :character   Median :0.000000
##                     Mean   :1989                      Mean   :0.001202
##                     3rd Qu.:1996                      3rd Qu.:0.000000
##                     Max.   :1998                      Max.   :1.000000
##                     NA's   :1
```

```
##       Action          Adventure       Animation       Children's
## Min.   :0.0000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
## 1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000
## Median :0.0000   Median :0.00000   Median :0.00000   Median :0.00000
## Mean   :0.1496   Mean   :0.07993   Mean   :0.02524   Mean   :0.07212
## 3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000
## Max.   :1.0000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000
##
##       Comedy          Crime           Documentary          Drama
## Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000
## 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.0000
## Median :0.0000   Median :0.0000   Median :0.00000   Median :0.0000
## Mean   :0.3017   Mean   :0.0643   Mean   :0.03005   Mean   :0.4303
## 3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:1.0000
## Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000
##
##       Fantasy         Film-Noir       Horror          Musical
## Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
## 1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000
## Median :0.00000   Median :0.00000   Median :0.00000   Median :0.00000
## Mean   :0.01322   Mean   :0.01442   Mean   :0.05409   Mean   :0.03365
## 3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000
## Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000
##
##       Mystery         Romance         Sci-Fi          Thriller
## Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.000
```

```
##  1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.000
##  Median :0.00000   Median :0.0000   Median :0.0000   Median :0.000
##  Mean   :0.03606   Mean   :0.1466   Mean   :0.0601   Mean   :0.149
##  3rd Qu.:0.00000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.000
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.000
##
##        War               Western
##  Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.00000   1st Qu.:0.00000
##  Median :0.00000   Median :0.00000
##  Mean   :0.04267   Mean   :0.01623
##  3rd Qu.:0.00000   3rd Qu.:0.00000
##  Max.   :1.00000   Max.   :1.00000
##
print(nratings(movie_ratings)) #Number of Total Ratings
## [1] 99392
print(rowCounts(movie_ratings[1:5,])) #First 5 users and their ratings
##   1   2   3   4   5
## 271  61  51  23 175
getRatingMatrix(movie_ratings[1:10, 1:5]) #Subset of Ratings Matrix of the first 10 users and 5 movies.
## 10 x 5 sparse Matrix of class "dgCMatrix"
##        Toy Story (1995) GoldenEye (1995) Four Rooms (1995) Get Shorty (1995)
## 1                 5                 3                 4                 3
## 2                 4                 .                 .                 .
## 3                 .         .       .                 .
## 4                 .         .       .                 .
```

```
## 5          4          3          .          .
## 6          4          .          .          .
## 7          .          .          .          5
## 8          .          .          .          .
## 9          .          .          .          .
## 10         4          .          .          4
##    Copycat (1995)
## 1          3
## 2          .
## 3          .
## 4          .
## 5          .
## 6          .
## 7          .
## 8          .
## 9          .
## 10         .
image(movie_ratings[1:20,1:20]) #Nice image of 20 by 20 subset of ratings matrix
as(movie_ratings[1, ], "list")[[1]][1:10] # The first 10 movies rated by user 1
##                    Toy Story (1995)
##                               5
##                    GoldenEye (1995)
##                               3
##                    Four Rooms (1995)
##                               4
##                    Get Shorty (1995)
```

```
##                                         3
##                                 Copycat (1995)
##                                         3
## Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)
##                                         5
##                                 Twelve Monkeys (1995)
##                                         4
##                                         Babe (1995)
##                                         1
##                                 Dead Man Walking (1995)
##                                         5
##                               Richard III (1995)
##                                         3
#In this chunk, we will get a summary of the ratings matrix and plot a bar plot of the ratings distribution
summary(getRatings(movie_ratings))
##        Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##        1.00    3.00    4.00    3.53    4.00    5.00
data.frame(ratings = getRatings(movie_ratings)) %>% ggplot(aes(ratings)) + geom_bar(width = 0.75, fill = "#FF6666") +  labs(title = 'Movies Ratings Distribution')
#Plotting the number of ratings per MovieLense reviewer.
summary(rowCounts(movie_ratings))
##        Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##        19.0    32.0    64.0   105.4   147.5   735.0
rowCounts(movie_ratings) %>%
data.frame(reviews_per_person = .) %>%
 ggplot(aes(x = reviews_per_person)) +
        geom_histogram(aes(y = ..density..), binwidth = 20, fill = "#42adf5") +
```

```
        scale_y_continuous(limits = c(0,.0125),
                           breaks = seq(0, .0125, by = 0.0025),
                           labels = seq(0, .0125, by = 0.0025)) +
        labs(title = 'Number of Ratings Per MovieLense Reviewer')
#Plotting the number of reviews per MovieLense listed movie.
summary(colCounts(movie_ratings))
##        Min. 1st Qu.  Median     Mean 3rd Qu.    Max.
##        1.00      7.00   27.00   59.73   80.00  583.00
colCounts(movie_ratings) %>%
 data.frame(movie_review_count = .) %>%
 ggplot(aes(x = movie_review_count)) +
        geom_histogram(aes(y = ..density..), binwidth = 20, fill = "#f5b642") +
        scale_y_continuous(limits = c(0,.0175)) +
        labs(title = 'Number of Reviews Per MovieLense listed Movie')
#Box plot of average rating per user.
summary(rowMeans(movie_ratings))
##        Min. 1st Qu.  Median     Mean 3rd Qu.    Max.
##   1.497   3.325   3.619   3.588   3.871   4.870
rowMeans(movie_ratings) %>%
 data.frame(mean_rating_per_user = .) %>%
 ggplot(aes(x = mean_rating_per_user)) +
        geom_boxplot(alpha=0.5, width=0.15, fill = "#00AFBB", color = "#00AFBB") +
        labs(title = 'Average Rating per User Boxplot')
#Box plot of average rating per movie.
summary(colMeans(movie_ratings))
##        Min. 1st Qu.  Median     Mean 3rd Qu.    Max.
```

```
##   1.000   2.665   3.162   3.077   3.653   5.000
colMeans(movie_ratings) %>%
 data.frame(mean_rating_per_movie = .) %>%
 ggplot(aes(x = mean_rating_per_movie)) +
        geom_boxplot(alpha=0.5, width=0.15,fill = "#FFDB6D", color = "#C4961A") +
        labs(title = 'Average Rating per Movie Boxplot')
#Looking at the user similarities between the first five users.
user_similarity <- similarity(movie_ratings[1:5, ],
                      method = "cosine",
                      which = "users")
as.matrix(user_similarity)
##        1         2      3         4         5
## 1 0.0000000 0.9605820 0.8339504 0.9192637 0.9326136
## 2 0.9605820 0.0000000 0.9268716 0.9370341 0.9848027
## 3 0.8339504 0.9268716 0.0000000 0.9130323 1.0000000
## 4 0.9192637 0.9370341 0.9130323 0.0000000 0.9946918
## 5 0.9326136 0.9848027 1.0000000 0.9946918 0.0000000
#Looking at the item (movie) similarities of the first five movies.
movie_similarity <- similarity(movie_ratings[, 1:5], method =
                               "cosine", which = "items")
as.matrix(movie_similarity)
##                    Toy Story (1995) GoldenEye (1995) Four Rooms (1995)
## Toy Story (1995)      0.0000000        0.9487374        0.9132997
## GoldenEye (1995)              0.9487374        0.0000000        0.9088797
## Four Rooms (1995)             0.9132997        0.9088797        0.0000000
## Get Shorty (1995)     0.9429069        0.9394926        0.8991940
```

```
## Copycat (1995)        0.9613638        0.9426876        0.9424719
##                  Get Shorty (1995) Copycat (1995)
## Toy Story (1995)      0.9429069        0.9613638
## GoldenEye (1995)              0.9394926        0.9426876
## Four Rooms (1995)             0.8991940        0.9424719
## Get Shorty (1995)     0.0000000        0.8919936
## Copycat (1995)              0.8919936   0.0000000
#Normalizing movie ratings before we fit the UBCF model.
norm_movie_ratings<- normalize(movie_ratings, method = 'center')
train_proportion <- .75 #Training data is 75% of the data set.
min(rowCounts(movie_ratings)) #Counting the minimum number of reviews by a reviewer.
## [1] 19
items_per_test_user_keep <- 10 #This will be the threshold for top N recommended movies. Should be >
than min(rowCounts(movie_ratings))
good_threshold <- 4 #For Binary classification
set.seed(123, sample.kind = "Rounding")
## Warning in set.seed(123, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
#Creating the training scheme for our model.
model_train_scheme <- movie_ratings %>%
 evaluationScheme(method = 'split', # Single train/test split
                  train = train_proportion, # Proportion of rows to train.
                  given = items_per_test_user_keep, # Threshold for top N recommended movies.
                  goodRating = good_threshold, #For Binary classification
                  k = 1)
#This will be the list of parameters for our model.
model_params <- list(method = "cosine",
```

```
                nn = 10, # Find each user's top 10 similar users
                sample = FALSE, #Already performed train/test split.
                normalize = "center")
model1 <- getData(model_train_scheme, "train") %>% #Fitting UBCF model to training set.
 Recommender(method = "UBCF", parameter = model_params)
model1_pred <- predict(model1, getData(model_train_scheme, "known"), type = "ratings") #Testing model on test set (new data)
model1_pred #Our new ratings matrix
## 236 x 1664 rating matrix of class 'realRatingMatrix' with 107143 ratings.
getRatingMatrix(model1_pred[1:10, 1:5]) #Snippet of new ratings matrix
## 10 x 5 sparse Matrix of class "dgCMatrix"
##      Toy Story (1995) GoldenEye (1995) Four Rooms (1995) Get Shorty (1995)
## 2    .          .                .          .
## 3    2.951470   1.987740         1.622451   0.9877396
## 4    4.186771   4.109677         .          4.2521360
## 10             4.670179         .          4.455319   .
## 12             3.994480         .          1.770588   3.4470588
## 13             4.536658         .          .          .
## 18             4.590910         .          3.698276   4.4196077
## 20             3.903520         .          .          .
## 25             4.888919         .          .          .
## 30             4.375947         .          .          .
##      Copycat (1995)
## 2    .
## 3    1.394834
## 4    .
## 10   .
```

```
## 12    .

## 13    .

## 18    .

## 20    2.618182

## 25    .

## 30    .
```

```
test_error <- calcPredictionAccuracy(model1_pred, getData(model_train_scheme, "unknown"), byUser = TRUE) #Calculating test error for each user.

head(test_error)  #Errors of test users.
```

```
##         RMSE              MSE     MAE

## 2  0.8510664 0.7243141 0.6868000

## 3  1.8498864 3.4220796 1.4407673

## 4  0.9682790 0.9375642 0.8370492

## 10 1.0683248 1.1413179 0.8676704

## 12 1.1338321 1.2855751 0.8716034

## 13 1.4519235 2.1080819 1.1518155
```

```
#Plotting distribution of RMSE on predicted recommendations per test user.

test_user_error_data <- data.frame(user_id = as.numeric(row.names(test_error)),
                                   rmse = test_error[, 1],
                                   predicted_items_cnt = rowCounts(getData(model_train_scheme, "unknown")))

test_user_error_data %>%
 ggplot(aes(rmse)) +
        geom_histogram(aes(y = ..density..), binwidth = 0.055, fill = "#e69191") +
        labs(title = 'RMSE on Predicted Recommendations per Test User',
        subtitle = "User-Based Collaborative Filtering using Cosine Similarity")

total_test_error <- calcPredictionAccuracy(model1_pred, getData(model_train_scheme, "unknown"), byUser = FALSE) #Calculating overall test error.
```

```
total_test_error
##       RMSE              MSE     MAE
## 1.2639355 1.5975330 0.9930037
#Getting the top 10 recommendations per test user.
top_recommendations <- 10
predicted_recommendations <- predict(object = model1,
                                     newdata =  getData(model_train_scheme, "known"),
                                    n = top_recommendations)
predicted_recommendations
## Recommendations as 'topNList' with n = 10 for 236 users.
#Displaying the top 10 recommendations for the first test user.
first_test_user <- predicted_recommendations@items[[1]]
movies_first_test_user <- predicted_recommendations@itemLabels[first_test_user]
movies_first_test_user
##  [1] "Mrs. Brown (Her Majesty, Mrs. Brown) (1997)"
##  [2] "Singin' in the Rain (1952)"
##  [3] "Dead Man Walking (1995)"
##  [4] "Ridicule (1996)"
##  [5] "Ulee's Gold (1997)"
##  [6] "Wings of the Dove, The (1997)"
##  [7] "Michael Collins (1996)"
##  [8] "Titanic (1997)"
##  [9] "Rear Window (1954)"
## [10] "Twelve Monkeys (1995)"
```