Data 612 - Project 3

Paul Perez

6/16/2020

Contents

The Recommender System's

This recommender system uses the `recommenderlab` package, and we'll load the MovieLense dataset. We can see that using the `class()` function, that this MovieLense dataset is a `realRatingMatrix`.

Data Ingestion, Selection, Manipulation

```
data("MovieLense")
class(MovieLense)
```

```
## [1] "realRatingMatrix"
## attr(,"package")
## [1] "recommenderlab"
```

```
matrix <- as(MovieLense, "realRatingMatrix")
```

Data Exploration

Using the `dim()` function, we can see the dimensions of the MovieLense matrix.

```
dim(matrix)
```

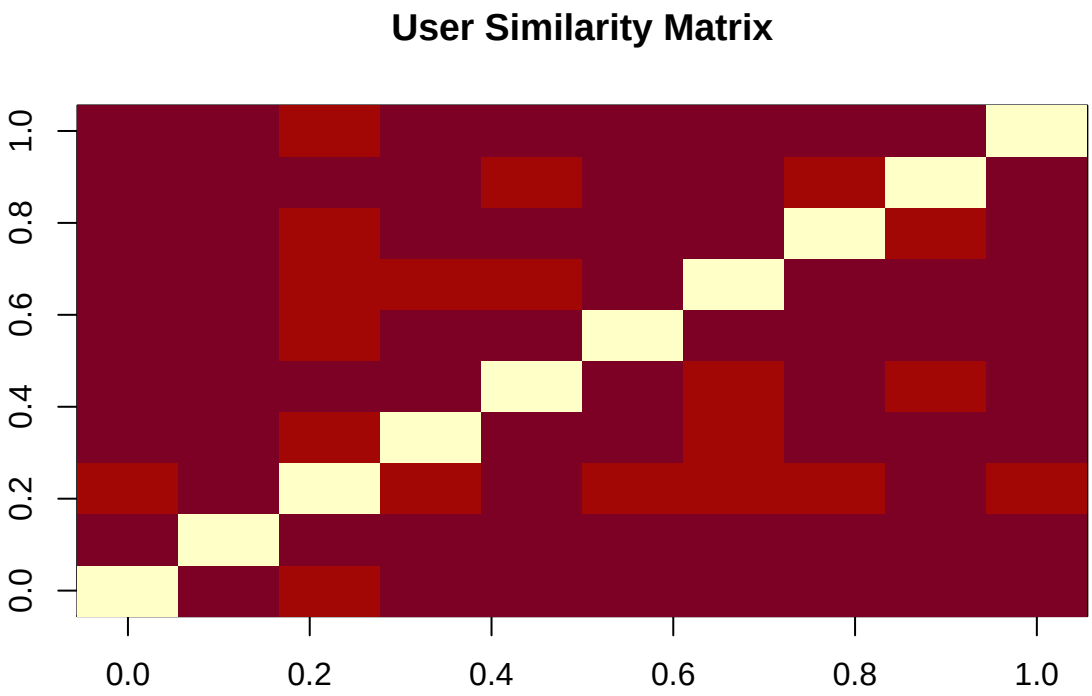
```
## [1]  943 1664
```

There are `943` users and `1664` movies.

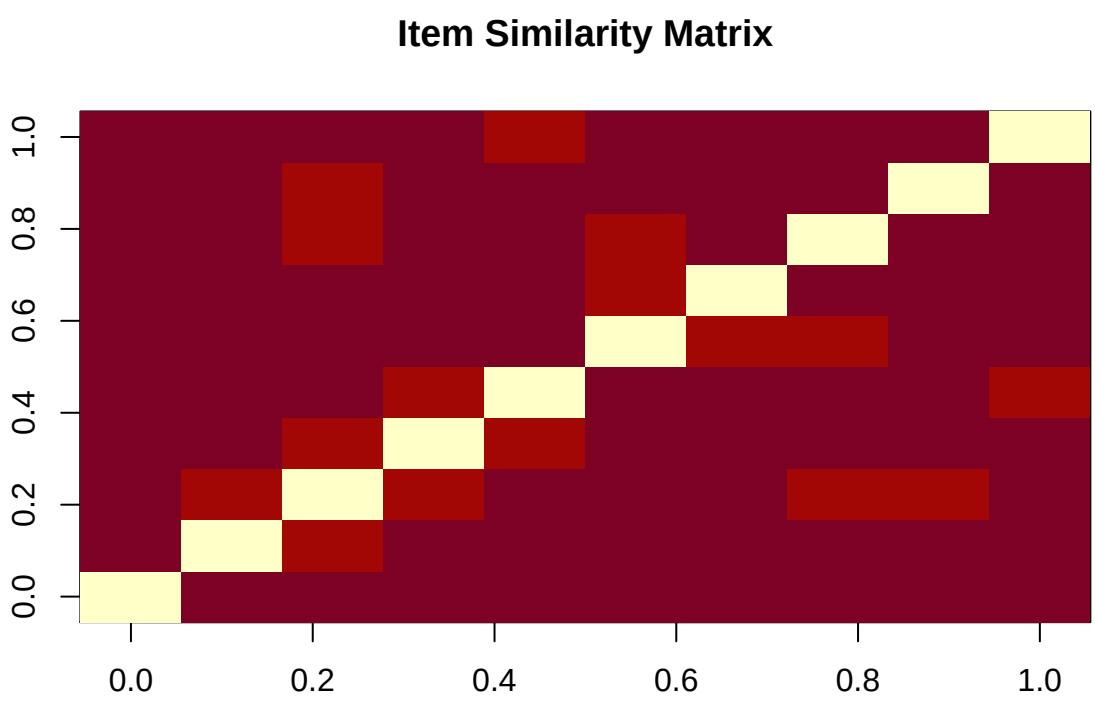
Similarity Matrix (User & Item)

The similarity matrix measures the similarity between item's or user's within the matrix. For the purpose of this project, we'll use the `cosine` method.

```
user_simularity <- similarity(matrix[1:10, ], method = "cosine", which = "users")
image(as.matrix(user_simularity), main = "User Similarity Matrix")
```




```r
item_similarity <- similarity(matrix[, 1:10], method = "cosine", which = "items")
image(as.matrix(item_similarity), main = "Item Similarity Matrix")
```



The darker the cell in each of these similarity matrices show's the greater similarity between the two user's or item's.

There are some emperical studies that show greater performance with the `cosine` similarity method for item based recommendation systems where the `pearson` similarity method shows greater performance for user based recommendation systems.

Since we loaded the data as a `realRatingMatrix`, which is an S4 class, we can look further into components of the object using `slotNames()`

```
slotNames(matrix)
```

```
## [1] "data"      "normalize"
```

By adding `@data` to the `matrix` object, we'll be able to take a look at this class, which happens to be a `dgCMatrix` class that inherits from the original matrix.

```
class(matrix@data)
```

```
## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"
```

By converting the ratings to a vector, we can count the number of unique ratings available.

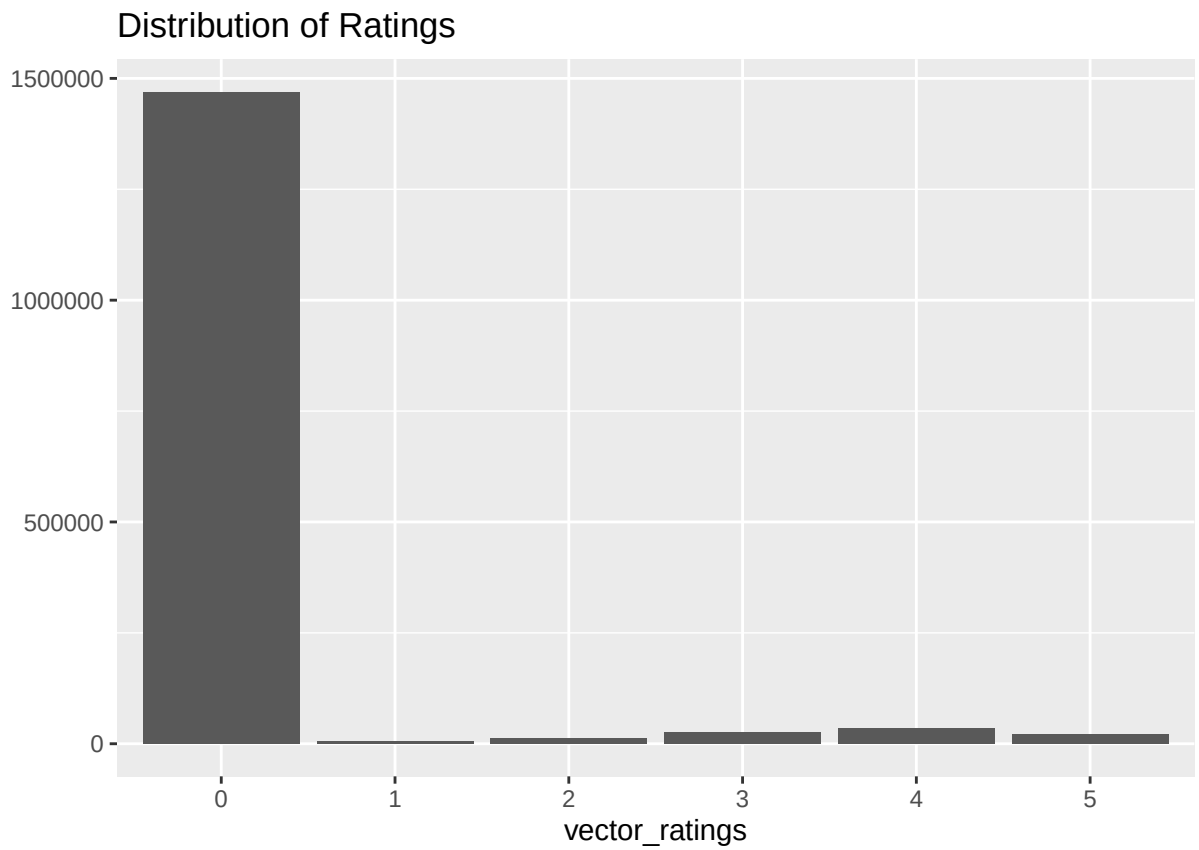
Table 1: Item Similarity Table

vector_ratings	Freq
0	1469760
1	6059
2	11307
3	27002
4	33947
5	21077

```
vector_ratings <- as.vector(matrix@data)
table_ratings <- table(vector_ratings)
table_ratings %>% kable(caption = "Item Similarity Table") %>% kable_styling("striped", full_width = TRU
```

We can look at the distribution of ratings.

```
vector_ratings <- factor(vector_ratings)
qplot(vector_ratings) + ggtitle("Distribution of Ratings")
```



Select Relevant Data

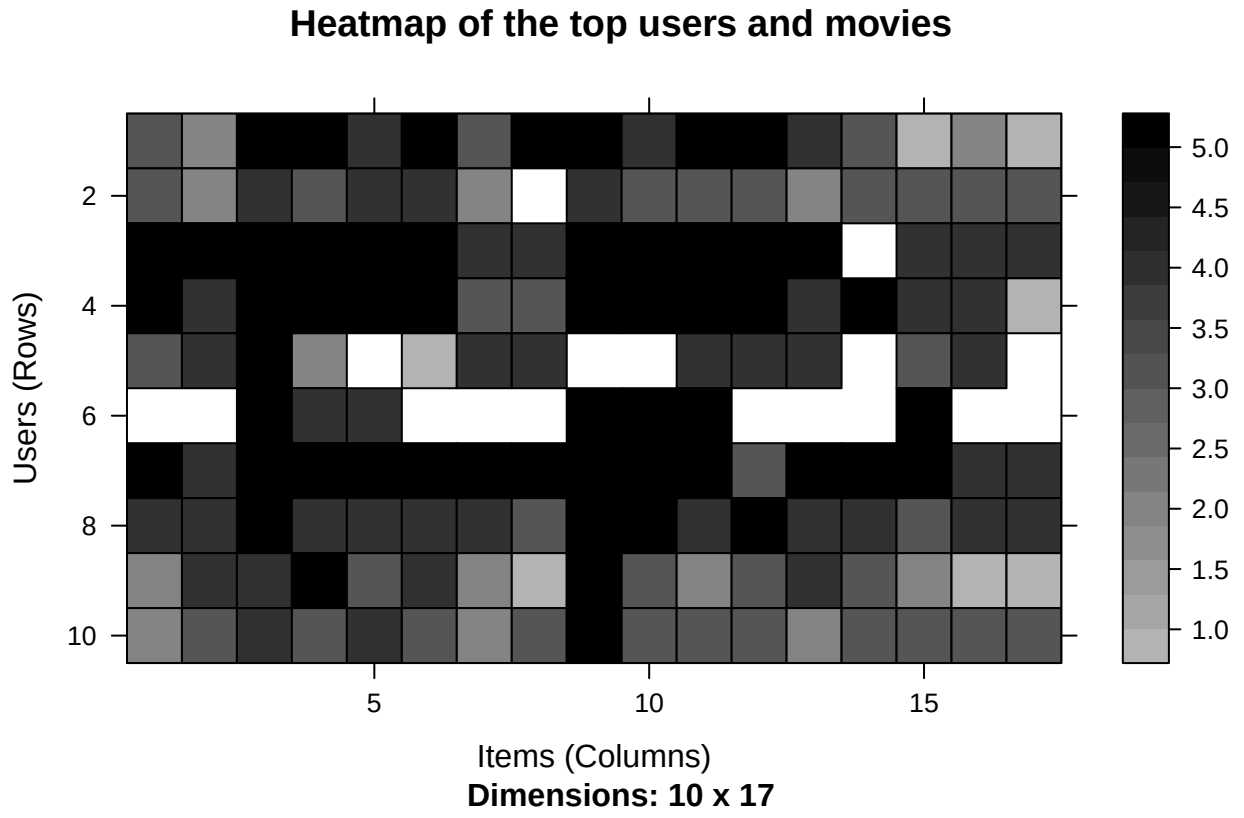
We can subset this data to select users who have rated at least 95 movies, and movies that have been watched at least 50 times. The dimension of the matrix changes from 943 x 1664 to 369 x 691

```r
movie_ratings <- matrix[rowCounts(matrix) > 95, colCounts(matrix) > 50]
dim(movie_ratings)
```

```
## [1] 369 591
```

Top 1% of Users

```r
min_movies <- quantile(rowCounts(matrix, na.rm = TRUE), 0.99)
min_users <- quantile(colCounts(matrix, na.rm = TRUE), 0.99)
image(matrix[rowCounts(matrix) > min_movies, colCounts(matrix) > min_users], main = "Heatmap of the top
```



Distribution of Average Rating by User

```r
average_rating <- rowMeans(movie_ratings)
qplot(average_rating) + stat_bin(binwidth = 0.2) + ggtitle ("Distribution of Average Rating by User")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

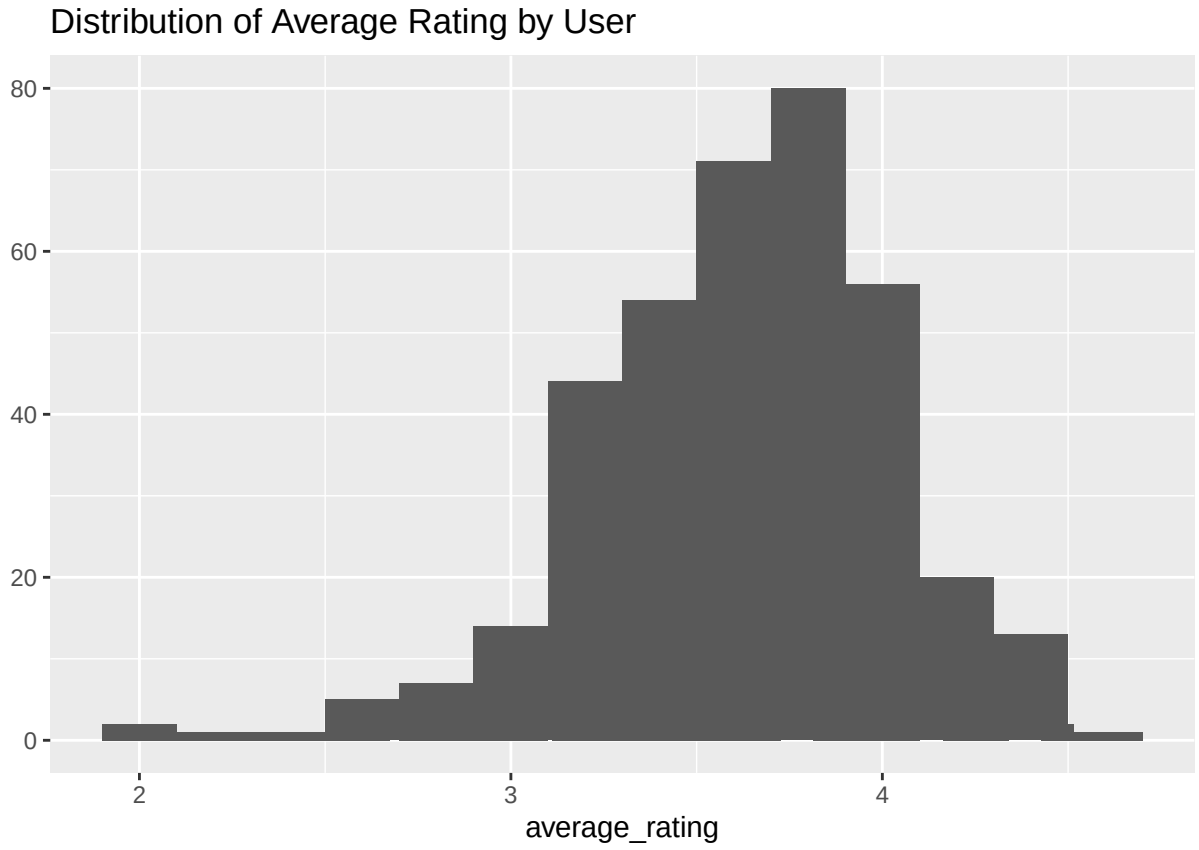

Distribution of Average Rating by User
80
60
40
20
0
2
3
4
average_rating

Table 2: User 1 Recommendations: Item-Item

x
Six Degrees of Separation (1993)
While You Were Sleeping (1995)
Man Who Knew Too Little, The (1997)
Othello (1995)
Space Jam (1996)

Split into Training & Test datasets using an 80/20 Ratio

```
set.seed(123)
train_sample <- sample(x = c(TRUE, FALSE), size = nrow(movie_ratings), replace = TRUE, prob = c(0.8, 0.2
movie_train <- movie_ratings[train_sample,]
movie_test <- movie_ratings[!train_sample,]
```

Item-Item Collaborative Filtering

This method identifies which items were similar in terms of other's items closely related.

```
system.time({
item_model <- Recommender(movie_train, method = "IBCF")
})
```

Create the Item-Item Collaborative Filter Recommender System with the training set

```
##    user  system elapsed
##    0.72    0.17    0.89
```

```
item_model
```

```
## Recommender of type 'IBCF' for 'realRatingMatrix'
## learned using 300 users.
```

```
item_predict <- predict(item_model, movie_test, n = 5)
```

Predict the Item-Item Collaborative Filter Recommender System on the test set Let's produce a list of the top 5 movie recommendations based on the Item-Item Collaborative Filter Recommender System.

```
item_reco_user_1 <- item_predict@items[[1]]
item_movie_user_1 <- item_predict@itemLabels[item_reco_user_1]
item_movie_user_1 %>% kable(caption = "User 1 Recommendations: Item-Item") %>% kable_styling("striped",
```

Table 3: User 1 Recommendations: User-User

x
Six Degrees of Separation (1993)
Wings of Desire (1987)
Titanic (1997)
Strictly Ballroom (1992)
Grand Day Out, A (1992)

User-User Collaborative Filtering

This method identifies similar user's to show close relations between the different users.

```
system.time({
user_model <- Recommender(movie_train, method = "UBCF")
})
```

```
##    user  system elapsed
##    0.01    0.00    0.02
```

```
user_model
```

```
## Recommender of type 'UBCF' for 'realRatingMatrix'
## learned using 300 users.
```

```
user_predict <- predict(user_model, movie_test, n = 5)
```

Predict the User-User Collaborative Filter Recommender System on the test set Let's produce a list of the top 5 movie recommendations based on the User-User Collaborative Filter Recommender System.

```
user_reco_user_1 <- user_predict@items[[1]]
user_movie_user_1 <- user_predict@itemLabels[user_reco_user_1]
user_movie_user_1 %>% kable(caption = "User 1 Recommendations: User-User") %>% kable_styling("striped",
```

Singular Vector Decomposition Collaborative Filtering

This method decomposes a matrix into arrays of different feature vectors corresponding to each column and row.

```
system.time({
svd_model <- Recommender(movie_train, method = "SVD")
})
```

Create the SVD Recommender System with the training set

Table 4: User 1 Recommendations: SVD

x
Six Degrees of Separation (1993)
While You Were Sleeping (1995)
Man Who Knew Too Little, The (1997)
Othello (1995)
Space Jam (1996)

```
##    user  system elapsed
##    0.03    0.00    0.03
```

```
svd_model
```

```
## Recommender of type 'SVD' for 'realRatingMatrix'
## learned using 300 users.
```

```
svd_predict <- predict(svd_model, movie_test, n = 5)
```

Predict the SVD Recommender System on the test set Let's produce a list of the top 5 movie recommendations based on the SVD Recommender System.

```
svd_reco_user_1 <- item_predict@items[[1]]
svd_movie_user_1 <- item_predict@itemLabels[svd_reco_user_1]
svd_movie_user_1 %>% kable(caption = "User 1 Recommendations: SVD") %>% kable_styling("striped", full_w
```

Summary

With the introduction of Singular Vector Decomposition, we had another method in which we could build a recommender system. Each of the three recommender system's support different theories, and take different computing power. We could see the efficiency of the SVD model compared to the Item and User models. The size of the data does affect performance, and the larger the data sets, the greater the focus on the model.

Source Code GitHub Repository