Hands-On Exercise: Implementing a Basic Recommender

In this Hands-On Exercise, you will build a movie recommendation engine. You will use both the Content-based Filtering approach and the Collaborative Filtering approach to build a basic movie recommendation engine.

The Dataset

The dataset used was from MovieLens, and is publicly available here (https://files.grouplens.org/datasets/movielens/ml-latest-small.zip).

In a bid to keep the recommender simple, we will use the smallest dataset available (ml-latest-small.zip)

Exercise #1:

1. Download, save and extract files
2. Note the location of where the files are. You will need the path shortly
3. Examine the files (movies & ratings) in Excel or spreadsheet program to get a sense of the file structures
4. You can delete the timestamp column in the ratings csv file
5. Open your R console
6. Explore the structure of the files

Before we get started, install the following libraries:

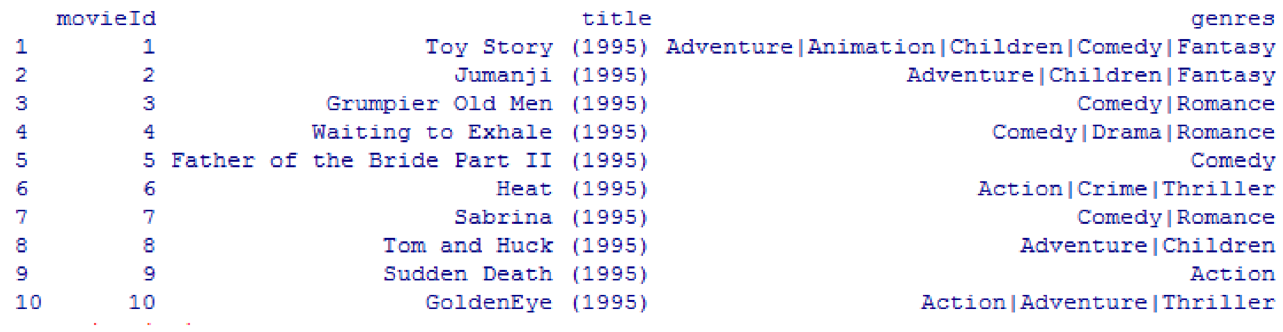
```
> install.packages("data.tables")
> install.packages("ggplot2")
> install.packages("recommenderlab")
```

Now, we are ready to START!

```
> movies = read.csv("path-to-your-movies.csv file") //load movies.csv file

> str(movies)//list the structure of movies
```

```
'data.frame':	9742 obs. of  3 variables:
 $ movieId: int  1 2 3 4 5 6 7 8 9 10 ...
 $ title  : Factor w/ 9123 levels "'burbs, The (1989)",..: 8301 4319 3421 8648 2762 3592 6860 8253 7673 3288 ...
 $ genres : Factor w/ 902 levels "(no genres listed)",..: 329 394 687 646 596 242 687 377 2 124 ...
```

```
> head(movies, n=10) //show first 10 rows of movie dataset
```



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

```
> ratings = read.csv("path-to-your-ratings.csv file") //load ratings.csv file

> str(ratings)//list the structure of movies
```

```
'data.frame':   100836 obs. of  4 variables:
 $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ movieId  : int  1 3 6 47 50 70 101 110 151 157 ...
 $ rating   : num  4 4 4 5 5 3 5 4 5 5 ...
 $ timestamp: int  1260759144 1260759179 1260759182 1260759185 1260759205 1260759151 1260759187 1260759148
1260759125 1260759131 ...
```

```
> head(ratings, n=10) //show first 10 rows of ratings dataset
```

```
   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
```

Exercise #2:

Yesterday, you did visualizations in R using ggplot. Now is your time to brag about your awesome visualization skills. Let us see which group creates some innovative visualizations from the ratings dataset.

7. Create a basic histogram with count on the y-axis and unique ratings on the x-axis
8. Now, wow us! – Do your thing, create an amazing visualization
9. Here below is a simple example to get you started!

```
> library(ggplot2)
> Plot <- ggplot(ratings, aes(x = rating)) + geom_histogram()
> plot
```

Content-based Filtering Approach

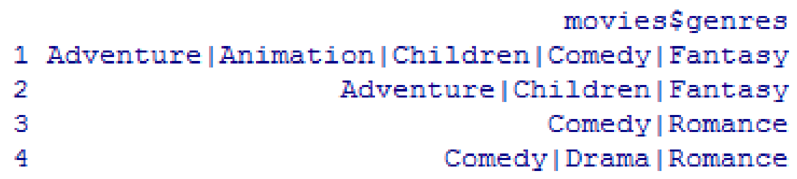
Like the name suggests, the Content-based Filtering approach involves analyzing an item a user interacted with and giving recommendations that are similar in content to that item. Content, in this case, refers to a set of attributes/features that describes your item. For a movie recommendation engine, a content-based approach would be to recommend movies that are of highest similarity based on its **features**, such as genres, actors, directors, year of production, etc. The assumption here is that users have preferences for a certain type of product, so we try to recommend a similar product to what the user has expressed liking for. Also, the goal here is to provide alternatives or substitutes to the item that was viewed.

We will be building a basic content-based recommender engine based on **movie genres** only.

Exercise #3:

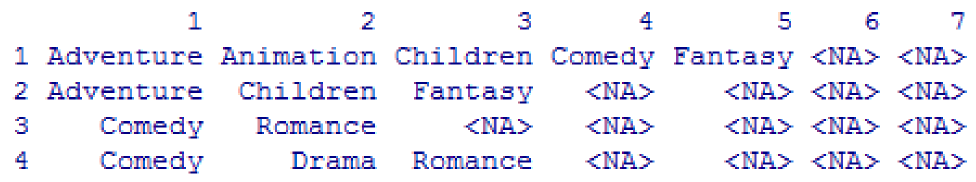
1. Obtain the movie features matrix by using the `tstrsplit()` function from the `data.table` package to split the pipe-separated genres available in the movies dataset. You may use the following steps:
   a. Create a data frame of basically `movies$genres` called "`movgen`"
   b. Leverage the `tstrsplit()` function to split pipe-separated genres and create a resulting data frame called "`movgen2`". `tstrsplit()` is a convenient wrapper function to split a column using `strsplit()` and assign the transposed result to individual columns.
   c. Column names should be labeled, 1 through 7
2. Your result should look like the table extract below

Before split: (movgen)



```
                              movies$genres
1 Adventure|Animation|Children|Comedy|Fantasy
2                Adventure|Children|Fantasy
3                            Comedy|Romance
4                      Comedy|Drama|Romance
```

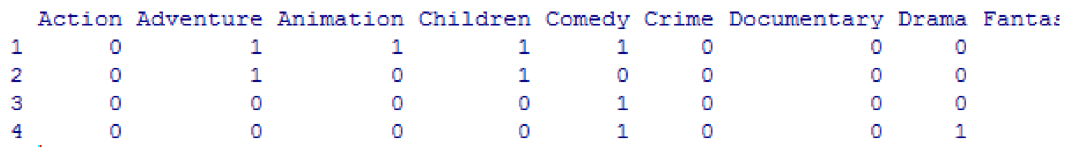
After Split: (movgen2)



```
          1         2        3      4       5    6    7
1 Adventure Animation Children Comedy Fantasy <NA> <NA>
2 Adventure  Children  Fantasy   <NA>    <NA> <NA> <NA>
3    Comedy   Romance     <NA>   <NA>    <NA> <NA> <NA>
4    Comedy     Drama  Romance   <NA>    <NA> <NA> <NA>
```

Exercise #4:

1. Create a matrix with columns representing every unique genre and indicate whether a genre was present or not in each movie.
2. Use the following 18 unique genres
   ```
   movgen_list <- c ("Action", "Adventure", "Animation",
   "Children", "Comedy", "Crime", "Documentary", "Drama",
   "Fantasy", "Film-Noir", "Horror", "Musical", "Mystery",
   "Romance", "Sci-Fi", "Thriller", "War", "Western")
   ```
3. You may use the following steps:
   a. Create an empty matrix "`movgen_matrix`" with 18 columns and (n+1) rows, where n=number of rows in the data set
   b. Set first row to genre list
   c. Set column names to genre list
   d. Iterate through `movgen_matrix` and populate with value 1, corresponding to associated genres in `movgen2`
   e. Convert to a data frame `movgen_matrix2`.
   f. Ensure you convert the characters to integers as you will perform some computations in the following exercises with the values in the matrix
4. Your result should look like the table extract below



```
  Action Adventure Animation Children Comedy Crime Documentary Drama Fanta:
1      0         1         1        1      1     0           0     0
2      0         1         0        1      0     0           0     0
3      0         0         0        0      1     0           0     0
4      0         0         0        0      1     0           0     1
```

We have now obtained the movie genres matrix. Each column represents a unique movie genre, and each row is a unique movie. We have 18 unique genres and n unique movies.

Exercise #5: Create a User Matrix

1. Convert the ratings dataset into a binary format "`binary_ratings`" to keep things simple. Ratings of 4 and 5 are mapped to 1, representing likes, and ratings of 3 and below are mapped to -1, representing dislikes.
2. Your result should look like the table extract below

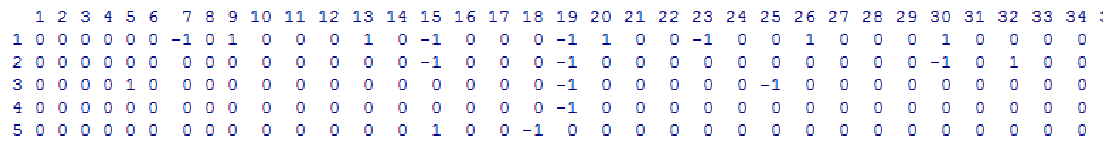
```
  userId movieId rating timestamp
1      1       1      1 964982703
2      1       3      1 964981247
3      1       6      1 964982224
4      1      47      1 964983815
5      1      50      1 964982931
6      1      70     -1 964982400
7      1     101      1 964980868
```

3. Use the dcast() function in the reshape2 package to transform the data from a long format to a wide format.

4. This also creates many NA values because not every user rated every movie. Substitute the NA values with 0. Use the code segment below accordingly

```
> binary_ratings2 <- reshape2::dcast(binary_ratings, movieId~userId, value.var = "rating", na.rm=
FALSE)

for (i in 1:ncol(binary_ratings2)){

  binary_ratings2[which(is.na(binary_ratings2[,i]) == TRUE),i] <- 0

}

> binary_ratings2 = binary_ratings2[,-1]#remove movieIds col. Rows are movieIds, cols are userIds

> dim(binary_ratings2)
```

5. The matrix has 9724 rows, representing the movieIds, and 610 columns, representing the userIds. The matrix should look like that shown below



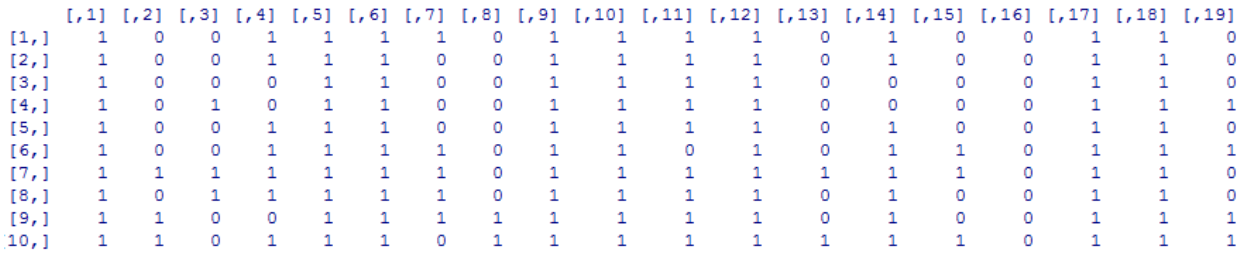
```
  1 2 3 4 5 6  7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 :
1 0 0 0 0 0 0 -1 0 1  0  0  0  1  0 -1  0  0  0 -1  1  0  0 -1  0  0  1  0  0  0  1  0  0  0  0
2 0 0 0 0 0 0  0 0 0  0  0  0  0  0 -1  0  0  0 -1  0  0  0  0  0  0  0  0  0  0 -1  0  1  0  0
3 0 0 0 0 1 0  0 0 0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  0
4 0 0 0 0 0 0  0 0 0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
5 0 0 0 0 0 0  0 0 0  0  0  0  0  0  1  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

6. You might notice that the movies dataset has 9742 movies, but the ratings dataset only has 9724 movies. To deal with this, remove the movies that have never been rated from the genres matrix. Use the code below:

```
> #Remove rows that are not rated from movies dataset

> unique_movieIds <- length(unique(movies$movieId)) #9742

> unique_ratings <- length(unique(ratings$movieId)) #9724

> movies2 <- movies[-which((unique_movieIds %in% unique_ratings) == FALSE),]

> rownames(movies2) <- NULL

> dim(movies2)

#Remove rows that are not rated from movgen_matrix2

> movgen_matrix3 <- movgen_matrix2[-which((unique_movieIds %in% unique_rating
s) == FALSE),]

> rownames(movgen_matrix3) <- NULL

> dim(movgen_matrix3)
```

7. Now we can calculate the dot product of the genre matrix (movgen_matrix3) and the ratings matrix (binary_ratings2) and obtain the user profiles. Ensure you convert to binary scale

8. The resulting User Profile should look like below



```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19]
 [1,]    1    0    0    1    1    1    1    0    1     1     1     1     0     1     0     0     1     1     0
 [2,]    1    0    0    1    1    1    0    0    1     1     1     1     0     1     0     0     1     1     0
 [3,]    1    0    0    0    1    1    0    0    1     1     1     1     0     0     0     0     1     1     0
 [4,]    1    0    1    0    1    1    0    0    1     1     1     1     0     0     0     0     1     1     1
 [5,]    1    0    0    1    1    1    0    0    1     1     1     1     0     1     0     0     1     1     0
 [6,]    1    0    0    1    1    1    1    0    1     1     0     1     0     1     1     0     1     1     1
 [7,]    1    1    1    1    1    1    1    0    1     1     1     1     1     1     1     0     1     1     0
 [8,]    1    0    1    1    1    1    1    0    1     1     1     1     0     1     0     0     1     1     0
 [9,]    1    1    0    0    1    1    1    1    1     1     1     1     0     1     0     0     1     1     1
 10,]    1    1    0    1    1    1    0    1    1     1     1     1     1     1     1     0     1     1     1
```

This user profiles shows the aggregated inclination of each user towards movie genres. Each column represents a unique userId, and positive values shows a preference towards a certain genre. The values were again simplified into a binary matrix — positive values were mapped to 1 to represent likes, negative values were mapped to 0 to represent dislikes.

Now that we have the user profiles, we can go 2 ways from here.

1. Predict if a user likes an item based on the item descriptions (movie genres). This can be done by predicting user movie ratings.
2. Assume that users like similar items and retrieve movies that are closest in similarity to a user's profile, which represents a user's preference for an item's feature.

Exercise #6: Create the Recommendation

1. Choose the second approach
2. Use Jaccard Distance to measure the similarity between user profiles, and the movie genre matrix. Consider only the ONE user in the dataset.
   HINT: Use the "proxy" package
3. What are your recommendation?

User-Based Collaborative Filtering Approach

The User-Based Collaborative Filtering approach groups users according to prior usage behavior or according to their preferences, and then recommends an item that a similar user in the same group viewed or liked. To put this in layman terms, if user 1 liked movie A, B and C, and if user 2 liked movie A and B, then movie C might make a good recommendation to user 2. The User-Based Collaborative Filtering approach mimics how word-of-mouth recommendations work in real life.

You will use User-Based Collaborative Filtering to generate a top-10 recommendation list for users using the recommenderlab package available in R. The recommenderlab package makes it easy to implement some of the popular collaborative filtering algorithms.

Exercise #7: Recommendation Models.

- Display the model applicable to the objects of type realRatingMatrix using recommenderRegistry$get_entries:

```
> library(recommenderlab)

> recommender_models <- recommenderRegistry$get_entries(dataType = "realRatin
gMatrix")

> names(recommender_models)
```

```
[1] "IBCF_realRatingMatrix"      "POPULAR_realRatingMatrix"    "RANDOM_realRatingMatrix"
"RERECOMMEND_realRatingMatrix" "SVD_realRatingMatrix"

[6] "SVDF_realRatingMatrix"      "UBCF_realRatingMatrix"
```

- Describe these models

```
> lapply(recommender_models, "[[", "description")
```

- We plan to use UBCF. Check the parameters of this model.

```
> recommender_models$UBCF_realRatingMatrix$parameters
```

- List the parameters of the model. What do they mean?

Exercise #8: Create Recommender Model.

Data Preprocessing

We need a ratings matrix to build a recommender model with recommenderlab. This can, again, be easily done with the dcast() function in the reshape2 package.

```
> library(reshape2)

> #Create ratings matrix. Rows = userId, Columns = movieId

rating_matrix <- dcast(ratings, userId~movieId, value.var = "rating", na.rm=F
ALSE)

> rating_matrix <- as.matrix(ratingmat[,-1]) #remove userIds
```

Creation of the Recommender Model

The User-based Collaborative Filtering recommender model was created with recommenderlab with the below parameters and the ratings matrix:

- Method: UBCF
- Similarity Calculation Method: Cosine Similarity
- Nearest Neighbors: 30

The predicted item ratings of the user will be derived from the 5 nearest neighbors in its neighborhood. When the predicted item ratings are obtained, the top 10 most highly predicted ratings will be returned as the recommendations.

1. Use the `dcast()` function in the `reshape2` package to create ratings matrix "`ubcf_ratingmatrix`" with rows = userId, columns = movieId
2. Use the parameters listed above for the UBCF recommender
3. Convert rating matrix (`rating_matrix`) into a recommenderlab sparse matrix
4. Normalize the data
5. Create Recommender Model. "UBCF" stands for User-Based Collaborative Filtering
6. Obtain recommendations