Recommender systems are one of the most lucrative examples of applied machine learning. Such techniques as cross-sell and up-sell are widely used by many e-commerce companies regardless of their products. This is a large subfield by itself, and today we will just approach the top of the iceberg.

We will download the dataset about movie recommendations: [https://grouplens.org/datasets/movielens/](https://grouplens.org/datasets/movielens/). Please, download `movies.csv` and `ratings.csv`. Then install the package `recommenderlab` that we use for the analysis.

```r
#install.packages("recommenderlab")
library(recommenderlab)
library(data.table)
library(tidy)
library(dplyr)
library(stringr)
library(ggplot2)

Setwd("your_path") # Set the path

# load the data
movies <- fread("movies.csv")
ratings <- fread("ratings.csv")
```

Again, first step is simple descriptive stats and visualizations:

```r
head(movies)
```

<table>
<thead>
<tr>
<th>##</th>
<th>movieId</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Toy Story (1995)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Jumanji (1995)</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Grumpier Old Men (1995)</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Waiting to Exhale (1995)</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>Father of the Bride Part II (1995)</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Heat (1995)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>##</th>
<th>genres</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Adventure</td>
</tr>
<tr>
<td>2</td>
<td>Adventure</td>
</tr>
<tr>
<td>3</td>
<td>Comedy</td>
</tr>
<tr>
<td>4</td>
<td>Comedy</td>
</tr>
<tr>
<td>5</td>
<td>Comedy</td>
</tr>
<tr>
<td>6</td>
<td>Action</td>
</tr>
</tbody>
</table>
Head(ratings)
userId movieId rating timestamp
1 31 2.5 1260759144
1 1029 3.0 1260759179
1 1061 3.0 1260759182
1 1129 2.0 1260759185
1 1172 4.0 1260759205
1 1263 2.0 1260759151

length(unique(movies$movieId))

## [1] 9125

dim(movies)
[1] 9125   3
#Thus each movie is present single time.

length(unique(ratings$userID)) # 671 users

## [1] 671

length(unique(ratings$movieId)) # 9066 movies

## [1] 9066

ggplot(ratings, aes(x=rating)) + geom_bar(stat='count', fill="#7ba367") + theme_bw()
ratings %>% group_by(userId) %>%
  summarise(avg_rating=mean(rating, na.rm=T)) %>%
  ggplot(aes(x=avg_rating)) +
  geom_histogram(fill="#7ba367", color='white') +
  theme_bw() +
  scale_x_continuous("average rating per user")

rm.rm: a logical value indicating whether NA values should be stripped before the computation proceeds.

Task 1(1pt): If there are any NA values in the ratings?
You probably paid attention that we have a timestamps. In case the same user rated the same movie several times, we aggregate the ratings:

```r
# user-ratings as a matrix
dim(ratings)

## [1] 100004  4
```

```r
ratings <- ratings %>%
  group_by(userId, movieId) %>%
  summarise(rating=mean(rating)) # in case the same user rated the same movie multiple times
dim(ratings)

## [1] 100004  3
```

As the dims the same, we could just use `ratings$timestamp <- NULL`

### Genre of the movie

Next, let’s try first to find movies that were similar based on their genres. As turned out, it is not as straightforward, as it seems :(. This is one *possible* solution, and not the most elegant one. You are free to use your own logic here.

```r
splitting_genres = strsplit(movies$genres, "\|", fixed=TRUE) # split vector genres by \\
                     
#class(splitting_genres) # splitting_genres is a list of characters. “Adventure”
                     
#fixed: logical. If TRUE match split exactly, otherwise use regular expressions.
genres <- unique(unlist(splitting_genres)) # collect all possible genres into one vector

# create matrix with zeros, where rows are movies and columns - genres
movie_genres_dummy <- as.data.frame(matrix(0, ncol=length(genres), nrow=nrow(movies)))
#Fill the matrix with 0
colnames(movie_genres_dummy) <- genres # assign names of genres to columns

# fill 1 if the genre is present for this movie, and 0 otherwise
# warning: may take longer time
for (movie_id in 1:length(splitting_genres)) {
  movie_genres_dummy[(movie_id, splitting_genres[[movie_id]]]) <- 1
}
```
head(movie_genres_dummy)

## Adventure Animation Children Comedy Fantasy Romance Drama Action Crime
## 1 1 1 1 1 1 0 0 0 0
## 2 1 0 1 0 1 0 0 0 0
## 3 0 0 0 1 0 1 0 0 0
## 4 0 0 0 1 0 1 1 0 0
## 5 0 0 0 1 0 0 0 0 0
## 6 0 0 0 0 0 0 0 0 1 1
## Thriller Horror Mystery Sci-Fi Documentary IMAX War Musical Western
## 1 0 0 0 0 0 0 0 0 0
## 2 0 0 0 0 0 0 0 0 0
## 3 0 0 0 0 0 0 0 0 0
## 4 0 0 0 0 0 0 0 0 0
## 5 0 0 0 0 0 0 0 0 0
## 6 1 0 0 0 0 0 0 0 0
## Film-Noir (no genres listed)
## 1 0
## 2 0
## 3 0
## 4 0
## 5 0
## 6 0

**Task 2 (1pt). Explain the matrix that we have. What does it show?**

We want to find similar movies. It can be used in such a way that if one client watches the movies, to recommend the similar in amount of genres. The most straightforward way is to calculate the distance (inverse of similarity). Note that we have a large amount of movies and calculating the distance for the whole matrix is computationally very expensive.

```r
#similarity between movies
dist(movie_genres_dummy[1,], movie_genres_dummy[2,], method = 'binary')
```

```
## 2
## 1 0.4
```

```r
dist(movie_genres_dummy[1,], movie_genres_dummy[5,], method = 'binary')
```

```
## 5
## 1 0.8
```
dist(movie_genres_dummy[1:10], method = 'binary')

##
## 1  2  3  4  5  6  7
## 2 0.400000
## 3 0.833333 1.000000
## 4 0.857143 1.000000 0.333333
## 5 1.000000 1.000000 0.000000 0.500000 0.666667
## 6 1.000000 1.000000 1.000000 1.000000
## 7 0.833333 1.000000 0.000000 0.333333 0.500000 1.000000
## 8 0.600000 0.333333 1.000000 1.000000 1.000000 1.000000 1.000000
## 9 1.000000 1.000000 1.000000 1.000000 1.000000 0.666667 1.000000
## 10 0.857143 0.800000 1.000000 1.000000 0.500000 1.000000
## 8 9
## 2
## 3
## 4
## 5
## 6
## 7
## 8
## 9 1.000000
## 10 0.750000 0.666667

min(dist(movie_genres_dummy[1:10], method = 'binary'))

## [1] 0

**Task 3 (1 pt):** How to find the row and column for the minimum value.

movie_genres_dummy[3,]
## Adventure Animation Children Comedy Fantasy Romance Drama Action Crime
## 3 0 0 0 0 1 0 0 0 0
## Thriller Horror Mystery Sci-Fi Documentary IMAX War Musical Western
## 3 0 0 0 0 0 0 0 0 0
## Film-Noir (no genres listed)
## 3 0 0

movie_genres_dummy[7,]

## Adventure Animation Children Comedy Fantasy Romance Drama Action Crime
## 7 0 0 0 0 1 0 0 0 0
## Thriller Horror Mystery Sci-Fi Documentary IMAX War Musical Western
## 7 0 0 0 0 0 0 0 0 0
## Film-Noir (no genres listed)
## 7 0 0
movies[c(3,7),]$title


Task 4 (2 pt). Try **two** different examples and find what are the names of similar movies. Does it make sense?

### Ratings of the movie

Next, we want to take into account ratings. For that task we will take advantage of **recommenderlab**, which requires matrix as an input. For more refer [1].

```r
ratings_spread <- spread(ratings, key=movieId, value=rating) # columns - movies, rows-users
#spread() takes two columns (key & value) and spreads in to multiple columns, it makes “long” data wider.
rating_matrix <- as.matrix(ratings_spread[,-1]) # exclude column with user ids
dimnames(rating_matrix) <- list(paste("u", unique(ratings$userId), sep=""),
                           paste("m", unique(ratings$movieId), sep=""))

rating_matrix_lab <- as(rating_matrix, "realRatingMatrix")
A matrix containing ratings (typically 1-5 stars, etc.)

# subset of the data
image(rating_matrix_lab[1:20,1:20])
#image(rating_matrix_lab) # too big to see
```
It is recommended to briefly take a look at this tutorial [2]: There are a lot of different recommender systems, listed by function. There are a lot of different recommender systems, listed by function

```r
recommenderRegistry$<get_entry_names()>
```

```r
## [1] "ALS_realRatingMatrix"    "ALSImplicit_realRatingMatrix"
## [3] "ALSimplicit_binaryRatingMatrix" "AR_binaryRatingMatrix"
## [5] "IBCF_binaryRatingMatrix"    "IBCFrealRatingMatrix"
## [7] "POPULAR_binaryRatingMatrix" "POPULARrealRatingMatrix"
## [9] "RANDOM_realRatingMatrix"    "RANDOM_binaryRatingMatrix"
## [13] "SVDF_realRatingMatrix"      "UBCF_binaryRatingMatrix"
## [15] "UBCFrealRatingMatrix"
```

```r
recommenderRegistry$<get_entry("POPULAR", dataType="realRatingMatrix")>
```

```r
## Recommender method: POPULAR for realRatingMatrix
## Description: Recommender based on item popularity.
## Reference: NA
## Parameters:
## normalize aggregationRatings aggregationPopularity
## 1 "center" new("standardGeneric" new("standardGeneric"
```

```r
recommenderRegistry$<get_entry("IBCF", dataType="realRatingMatrix")>
```

```r
## Recommender method: IBCF for realRatingMatrix
## Description: Recommender based on item-based collaborative filtering.
## Reference: NA
## Parameters:
## k method normalize normalize_sim_matrix alpha na_as_zero
## 1 30 "Cosine" "center" FALSE FALSE 0.5 FALSE
```

Let’s try our first model, which is recommendation based on popularity. The method computes average rating for each item based on available ratings and predicts each unknown rating as average for the item.

```r
model <- Recommender(rating_matrix_lab, method = "POPULAR")
recom <- predict(model, rating_matrix_lab[1:4], n=5)
as(recom, "list")
```

```r
## $u1
## [1] "m2734" "m2263" "m335" "m4019" "m2005"
##
## $u2
## [1] "m2734" "m335" "m2005" "m2108" "m2726"
```
## $u3
## [1] "m335"  "m2005"  "m292"  "m2108"  "m2726"
## $u4
## [1] "m2734"  "m4019"  "m152"  "m292"  "m2108"

prediction <- predict(model, rating_matrix_lab[1:5], type="ratings")
as(prediction, "matrix")[,1:5]

##
## m31 m1029 m1061 m1129 m1172
## u1 2.775976 2.394019 2.128042 1.301898 2.227099
## u2 3.712818 3.330861 3.064885 2.238740 3.163941
## u3 3.794603 3.412646 3.146670 2.320525 3.245727
## u4 4.574015 4.192058 3.926082 3.099937 4.025138
## u5 4.135976 3.754019 NA 2.661898 3.587099

### Evaluation of the recommender system

```
set.seed(5864)
eval_scheme <- evaluationScheme(rating_matrix_lab, method="split", train=0.8, given=-5)
# 5 ratings of 20% of users are excluded for testing

model_popular <- Recommender(getData(eval_scheme, "train"), "POPULAR")
prediction_popular <- predict(model_popular, getData(eval_scheme, "known"), type="ratings")

rmse_popular <- calcPredictionAccuracy(prediction_popular, getData(eval_scheme, "unknown"))[1]
```

```
## RMSE
## 0.935082
```

If you recall, RMSE (root mean square error) was also used in regression problems. It does not tell us much alone, but we can compare models:

```
# learn about input parameters via help
model_ubcf <- Recommender(getData(eval_scheme, "train"), "UBCF",
                           param=list(normalize = "center", method="Cosine", nn=50))
```

Note that we use the same scheme:

```
prediction_ubcf <- predict(model_ubcf, getData(eval_scheme, "known"), type="ratings")
```

```
rmse_ubcf <- calcPredictionAccuracy(prediction_ubcf, getData(eval_scheme, "unknown"))[1]
```

```
rmse_ubcf
```

```
```
rmse

rbind(calcPredictionAccuracy(prediction_popular, getData(eval_scheme, "unknown")),
      calcPredictionAccuracy(prediction_ubcf, getData(eval_scheme, "unknown")))

## RMSE      MSE        MAE
## [1,] 0.9350820 0.8743784 0.7000101
## [2,] 0.9682466 0.9375014 0.7423649

References:

[1] https://cran.r-project.org/web/packages/recommenderlab/recommenderlab.pdf