Collaborative Filtering Recommender Based on Co-occurrence Matrix

Author: Marjan Sterjev

Today we are facing recommendations everywhere. The social networks and Internet portals recommend products, movies, books, events, news... Recommendations are the essence of the marketing campaigns. These recommendations are not passive and user behavior agnostic. There are a lot of recommendation systems out there that recommend items to the users based on the user's preferences as well as user's past history (purchasing, viewing, rating etc.)

Collaborative Filtering is one the most exploited item recommendation approaches. It is not based on the item (movie, book..) content (genre, author, keywords …). Rather, it is based on the user's past history like buying, viewing, rating items.

Let's say we have users and movies. Each user in the system has watched one or more movies in the past. We will assume in this article that if the user has watched particular movie, he likes and prefers that movie. User to movie preferences can be represented as matrix model where each row represents particular user and each column represents particular movie. Value of 1 in a matrix cell means that the user has watched and prefers that movie.

In **R** we can generate sample user movie preference matrix the following way:

```
> set.seed(1)
> no_users <- 15
> no_movies <- 10
> user_ids <- apply(array(1:no_users),1,function(i)paste(c('u',i),collapse='_'))
> movie_ids <- apply(array(1:no_movies),1,function(i)paste(c('m',i),collapse='_'))
> m_user_movie <- matrix(
sample(x=c(0,1),size=no_users*no_movies,replace=TRUE,prob=c(0.7,0.3)),
ncol=no_movies,byrow=TRUE,dimnames=list(user_ids,movie_ids))
```

The generated matrix for 15 users and 10 movies is:

```
     m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8 m_9 m_10
u_1    0   0   0   1   0   1   1   0   0    0
u_2    0   0   0   0   1   0   1   1   0    1
u_3    1   0   0   0   0   0   0   0   1    0
u_4    0   0   0   0   1   0   1   0   1    0
u_5    1   0   1   0   0   1   0   0   1    0
u_6    0   1   0   0   0   0   0   0   0    0
u_7    1   0   0   0   0   0   0   1   0    1
u_8    0   1   0   0   0   1   1   0   1    1
u_9    0   1   0   0   1   0   1   0   0    0
u_10   0   0   0   1   1   1   0   0   1    0
u_11   0   0   0   1   0   0   0   0   1    0
u_12   1   1   0   0   0   0   1   0   0    0
u_13   1   0   0   0   1   0   0   0   0    0
u_14   0   0   0   0   1   0   0   0   1    0
u_15   0   0   0   0   1   0   0   1   0    1
```

So user 'u_1' has already watched movies 'm_4', 'm_6' and 'm_7', user 'u_2' has already watched movies 'm_5', 'm_7', 'm_8','m_10' and so on.

An interesting aspect in the matrix model above is finding how many times two movies have appeared together in the user preference vectors? For example, we can see that movies 'm_1' and 'm_2' appear together only once in the movie preference vector for user 'u_12'. However, movies 'm_5' and 'm_7' appear together three times in the user movie preference vectors for the users 'u_2', 'u_4' and 'u_9'.

```
     m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8 m_9 m_10
u_1    0   0   0   1   0   1   1   0   0    0
u_2    0   0   0   0   1   0   1   1   0    1
u_3    1   0   0   0   0   0   0   0   1    0
u_4    0   0   0   0   1   0   1   0   1    0
u_5    1   0   1   0   0   1   0   0   1    0
u_6    0   1   0   0   0   0   0   0   0    0
u_7    1   0   0   0   0   0   0   1   0    1
u_8    0   1   0   0   0   1   1   0   1    1
u_9    0   1   0   0   1   0   1   0   0    0
u_10   0   0   0   1   1   1   0   0   1    0
u_11   0   0   0   1   0   0   0   0   1    0
u_12   1   1   0   0   0   0   1   0   0    0
u_13   1   0   0   0   1   0   0   0   0    0
u_14   0   0   0   0   1   0   0   0   1    0
u_15   0   0   0   0   1   0   0   1   0    1
```

We can obtain the full co-occurrence matrix if we transpose the user movie preference matrix and calculate cross product between this transposed matrix and the original one.

```
> co_movie <- t(m_user_movie) %*% m_user_movie
```

```
     m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8 m_9 m_10
m_1    5   1   1   0   1   1   1   1   2    1
m_2    1   4   0   0   1   1   3   0   1    1
m_3    1   0   1   0   0   1   0   0   1    0
m_4    0   0   0   3   1   2   1   0   2    0
m_5    1   1   0   1   7   1   3   2   3    2
m_6    1   1   1   2   1   4   2   0   3    1
m_7    1   3   0   1   3   2   6   1   2    2
m_8    1   0   0   0   2   0   1   3   0    3
m_9    2   1   1   2   3   3   2   0   7    1
m_10   1   1   0   0   2   1   2   3   1    4
```

Diagonal elements of the co-occurrence matrix shall be set to zero because we are not interested how many times a movie appears with itself in the user preference vectors.

```
> diag(co_movie) <- 0
```

The final co-occurrence matrix is:

```
     m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8 m_9 m_10
m_1    0   1   1   0   1   1   1   1   2    1
m_2    1   0   0   0   1   1   3   0   1    1
m_3    1   0   0   0   0   1   0   0   1    0
m_4    0   0   0   0   1   2   1   0   2    0
```

```
m_5     1    1    0    1    0    1    3    2    3     2
m_6     1    1    1    2    1    0    2    0    3     1
m_7     1    3    0    1    3    2    0    1    2     2
m_8     1    0    0    0    2    0    1    0    0     3
m_9     2    1    1    2    3    3    2    0    0     1
m_10    1    1    0    0    2    1    2    3    1     0
```

Recommendations for particular user can be generated if we calculate matrix product between the co-occurrence matrix and that user's movie preference vector. I will explain below the idea behind this cross product. For example the user 'u_1' has the following movie preference vector:

```
> m_user_movie[1,]

 m_1  m_2  m_3  m_4  m_5  m_6  m_7  m_8  m_9 m_10
   0    0    0    1    0    1    1    0    0    0
```

The recommendation vector can be calculated as:

```
> co_movie %*% m_user_movie[1,]
     [,1]
m_1     2
m_2     4
m_3     1
m_4     3
m_5     5
m_6     4
m_7     3
m_8     1
m_9     7
m_10    3
```

What are the weighting numbers obtained in the recommendation vector? For each movie in the recommendation vector it is the number of times that movie appeared together with the movies contained in the user preference vector. In the example above, the weighting value for movie 'm_1' is 2 because it appeared together once with 'm_6' and once with 'm_7'; the value for movie 'm_2' is 4 because this movie appeared together once with 'm_6' and three times with 'm_7'; the value for movie 'm_9' is 7 because this movie appeared together twice with 'm_4', three times with 'm_6' and twice with 'm_7'.

In the movie recommendation vector we shall set to zero all movies already preferred by the user:

```
> co_movie %*% m_user_movie[1,] * (1-m_user_movie[1,])
     [,1]
m_1     2
m_2     4
m_3     1
m_4     0
m_5     5
m_6     0
m_7     0
m_8     1
m_9     7
m_10    3
```

The movie with the maximum weighting value in the recommendation vector is the one that shall be recommended to the user. In our example, user 'u_1' will be recommended the movie 'm_9' which has the highest weighting value of 7.

The steps explained above can be wrapped into recommendation function:

```
> recommend <- function(co_movie, a_user_movie){
      a_rec <- co_movie %*% a_user_movie
      a_rec <- a_rec * (1-a_user_movie)
      max_movie_idx <- which.max(a_rec)
      movie_id=paste(c('m',max_movie_idx),collapse='_')
      return(movie_id)
}

> recommend(co_movie, m_user_movie[1,])
[1] "m_9"

> recommend(co_movie, m_user_movie[15,])
[1] "m_7"
```

Sparse Matrices and Vectors

Matrices and Vectors where most of the elements are zero are **sparse**. There are a lot of proposed approaches that describe how to represent sparse structures in memory and how to implement some linear algebra operations like cross product between matrix and sparse vector.

Each user movie preference vector in the example above is sparse vector. Zeros appear in these vectors with probability of 0.7 and ones with probability of 0.3.

The weight for the movie 'm_1' in the recommendation vector above is obtained the following way:

```
0*0+1*0+1*0+0*1+1*0+1*1+1*1+1*0+2*0+1*0=1*1+1*1=2
```

The weight for the movie 'm_2' in the recommendation vector above is obtained as:

```
1*0+0*0+0*0+0*1+1*0+1*1+3*1+0*0+1*0+1*0=1*1+3*1=4
```

The total number of operations for the whole recommendation vector is 10 * (10 multiplications+9 additions)=190. Most of these 190 operations are not needed. There is a simple trick that can greatly reduce the number of the operations involved. First we need to find the positions in the user movie preference vector that contain 1's. For user 'u_1' these positions are:

```
> which(m_user_movie[1,]==1)
m_4 m_6 m_7
  4   6   7
```

The recommendation vector can be obtained if we sum the 4-th, 6-th and 7-th column from the co-occurrence matrix:

```
> co_movie[,4]+co_movie[,6]+co_movie[,7]
m_1  m_2  m_3  m_4  m_5  m_6  m_7  m_8  m_9 m_10
   2    4    1    3    5    4    3    1    7    3
```

The result is the same as the result obtained with the basic matrix product rule. However the number of operations is reduced to only 20 additions.

This approach can be expressed in a more compact form that results into this recommender function:

```
> recommend_1 <- function(co_movie, a_user_movie){
      a_rec <- rowSums(co_movie[,which(a_user_movie>0)])
      a_rec <- a_rec * (1-a_user_movie)
      max_movie_idx <- which.max(a_rec)
      movie_id=paste(c('m',max_movie_idx),collapse='_')
      return(movie_id)
}

> recommend_1(co_movie,m_user_movie[1,])
[1] "m_9"
```

This kind of linear algebra optimization is very important and essential when dealing with Big Data. In reality the systems contain thousands or millions of items, sparsity of the matrices is very large and the optimization presented in this article will drive you toward feasible system and optimal performance.