Movie Collaborative Filtering System Based on Low-rank Matrix Factorization

Zhu Zhengtian

```
Sys.setenv(LANGUAGE = "en")
```

Abstract

Recommendation systems, also called recommendation engines, are widely used in our daily life. Collaborative filtering systems, one of the recommendation systems, are used to predict future ratings of items by users based on their past ratings. GroupLens is one such collaborative filtering system. It uses the correlation between users for predictive purposes. However, the estimated correlation matrix sometimes has a serious defect: although the correaltion matrix is originally positive semidefinite, the estimated one may not be positive semidefinite when not all ratings are observed. To obtain a positive semidefinite correlation matrix, the nearest correlation matrix problem has been recently studied in the fields of numerical analysis and optimization. Since the optimization problem has a prescribed rank and bound constraints on the correlations, it can be converted into a orthogonal constrained optimization problem. In this project, we use a first-order feasible method to estimate the correlation matrix. We replace the original correlation matrix in the collaborative filtering system with our adjusted correlation matrix. We compare the efficieny between different recommendation algorithms.

Recommendation Engines

A recommendation engine uses data filtering algorithms to suggest content, offers, and products based on individual or audience profiles. It does this by using collaborative, content-based, or popularity-based rules to surface recommendations.

Content-based Engine

Content-based engine uses item features to recommend other items similar to what the user likes, based on their previous actions or explicit feedback. It does not require the opinion (rating) of one user. However, it needs plenty of information of user's action and precise description of item.

Popularity-based Engine

Popularity-based engine offers a very primitive form of collaborative filtering, where items are recommended to users based on how popular those items are among other users. This system does not require the opinion, even the previous action of one user. It only collects and recommends the most popular item.

Collaborative Filtering Engine

Collaborative filtering is commonly used for recommender systems. This technique aims to fill in the missing entries of a user-item association matrix. There are two main kinds of collaborative filtering systems, model-based collaborative fltering system and memory-based collaborative fltering system. Model-based systems

use item-item similarity, while memory-based systems use user-user similarity. Different from the above two engines, collaborative filtering engine requires the opinion (rating) of one user.

MovieLens

GroupLens

GroupLens is a research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities specializing in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems.

MovieLens Data

GroupLens Research has collected and made available rating data sets from the MovieLens web site (http://movielens.org). The data sets were collected over various periods of time, depending on the size of the set. In this project, we use MovieLens 100K dataset which can be downloaded at https://grouplens.org/datasets/movielens/100k/

MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota. This data set consists of:

- 100,000 ratings (1-5) from 943 users on 1682 movies.
- Each user has rated at least 20 movies.
- Simple demographic info for the users (age, gender, occupation, zip)

The data was collected through the MovieLens web site (movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998. This data has been cleaned up - users who had less than 20 ratings or did not have complete demographic information were removed from this data set.

The data can be achieved by loading package "recommenderlab". We also load other needed packages.

```
library(recommenderlab)
```

```
## Warning: package 'recommenderlab' was built under R version 3.6.3
## Loading required package: Matrix
## Loading required package: arules
## Warning: package 'arules' was built under R version 3.6.3
##
## Attaching package: 'arules'
## The following objects are masked from 'package:base':
##
##     abbreviate, write
## Loading required package: proxy
## Warning: package 'proxy' was built under R version 3.6.3
##
## Attaching package: 'proxy'
## The following object is masked from 'package:Matrix':
##
##     as.matrix
```

```
## The following objects are masked from 'package:stats':
##
##     as.dist, dist
```

```
## The following object is masked from 'package:base':
##
##     as.matrix
```

```
## Loading required package: registry
```

```
## Registered S3 methods overwritten by 'registry':
##   method               from
##   print.registry_field proxy
##   print.registry_entry proxy
```

```
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 3.6.3
```

```
library(countrycode)
library(ggplot2)
```

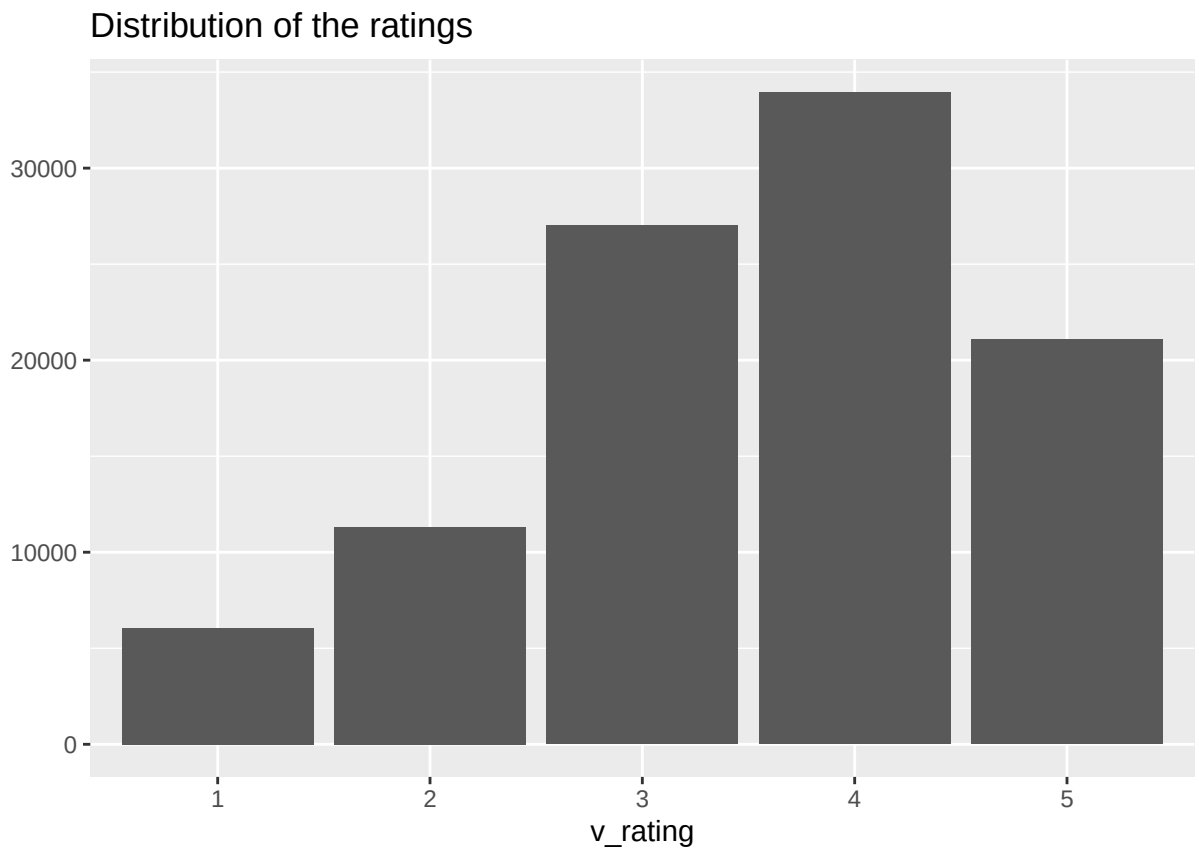
We then check the dimension of MovieLens data.

```
data("MovieLense")
dim(MovieLense)
```

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

We plot the distribution of ratings by users. From the plot, we can see the distribution is normal and reasonable.

```
v_rating <- as.vector(MovieLense@data)
v_rating <- v_rating[v_rating!=0]
v_rating <- factor(v_rating)
dis_plot <- qplot(v_rating) + ggtitle('Distribution of the ratings')
dis_plot
```



Algorithm Used in GroupLens

Overview of the Collaborative Filtering Process

The goal of a collaborative filtering algorithm is to suggest new items or to predict the utility of a certain item for a particular user based on the user's previous likings and the opinions of other like-minded users. In a typical CF scenario, there is a list of m users $U = \{u_1, u_2, \ldots, u_m\}$ and a list of n items $I = \{i_1, i_2, \ldots, i_n\}$ Each user u_i has a list of items I_{ui} , which the user has expressed his/her opinions about. Opinions can be explicitly given by the user as a rating score. Recommendation is a list of N items, $I_r \subset I$, that the active user will like the most. Note that the recommended list must be on items not already purchased by the active user, i.e.,This interface of CF algorithms is also known as **Top-N** recommendation.

GroupLens' Collaborative Filtering Algorithm

The algorithm used in GroupLens is explained in the following. Let R_{ia} be the rating of item a by user i and U_{ij} be the set of items rated by both user i and user j. The correlation coefficient between user i and user j is estimated by

$$\widehat{\rho}_{ij} = \frac{\sum_{a \in U_{ij}} (R_{ia} - \bar{R}_i(U_{ij}))(R_{ja} - \bar{R}_j(U_{ij}))}{\sqrt{\sum_{a \in U_{ij}} (R_{ia} - \bar{R}_i(U_{ij}))^2} \sqrt{\sum_{a \in U_{ij}} (R_{ja} - \bar{R}_j(U_{ij}))^2}}.$$

Here, $\bar{R}_i(U_{ij}) = \sum_{a \in U_{ij}} R_{ia}/n_{ij}$ is the mean of R_{ia} in U_{ij}, and n_{ij} is the number of elements in U_{ij}.

Let T_a be the set of users who rated item a and T_i' be the set of items rated by user i. The point prediction

for the rating of item a by user i is given by

$$\widehat{R}_{ia} = \bar{R}_i + \frac{\sum_{j \in T_a} \widehat{\rho}_{ij}(R_{ja} - \bar{R}_j)}{\sum_{j \in T_a} |\widehat{\rho}_{ij}|}.$$

where $\bar{R}_i$ is the mean of R_{ia} in T_i'.

Problems and Solutions

Positive Semidefiniteness

From the above algorithm, we can see that the key step is to estimate the correlation matrix. The positive semidefiniteness of the correlation matrix is not required in the above algorithm, but the correlation matrices are originally positive semidefinite. Thus, the existence of large negative eigenvalues suggests that $\hat{\rho}$ has a lot of noise.

We first try a simple collaborative filtering algorithm. We the split the total data and 80% of the data are training set, while the other 20% are test set.

```
trainset <- sample(x=c(T,F),size=nrow(MovieLense),replace=T,prob=c(0.8,0.2))
data_train <- MovieLense[trainset, ]
data_test <- MovieLense[!trainset, ]
```

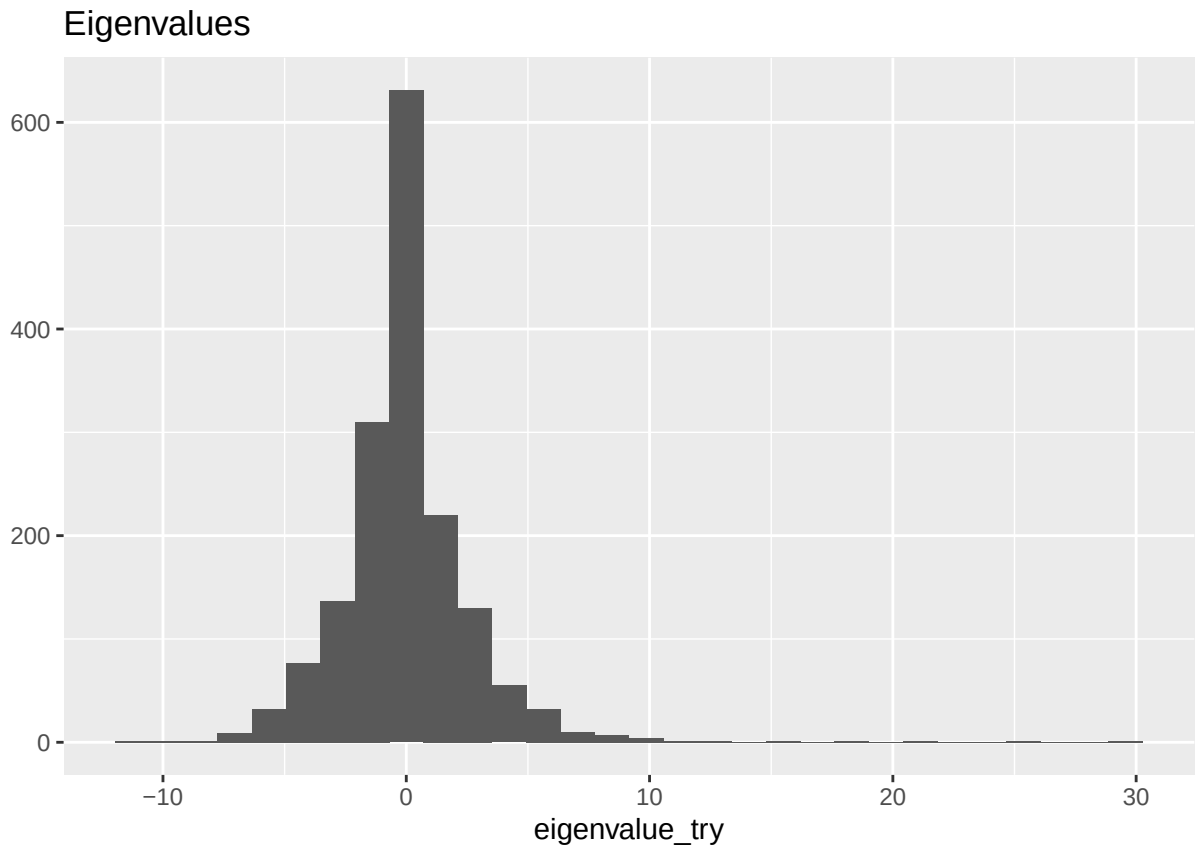
We choose the default IBCF (item-based collaborative filtering) algorithm.

```
model_try = Recommender(data = data_train,method="IBCF")
model_try_detail <- getModel(model_try)
```

We then check the similarity matrix. We can see that the eigenvalues of the similarity matrix have many negative values, which means the matrix has a lot of noises.

```
sim <- model_try_detail$sim
eigenvalue_try <- eigen(sim)$values
eigenvalue_try <- as.numeric(eigenvalue_try)
eigenvalue_plot1 <- qplot(eigenvalue_try) + ggtitle('Eigenvalues')
eigenvalue_plot1
```

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



Low-rank Matrix Factorization

Our goal is to find a low-rank positive semidefinite matrix of a given symmetric matrix. A low-rank matrix optimization problem is to be solved. This kind of problem belongs to semidefinite programming (SDP) which is a useful tool for modeling many applications.

Let S^n and S^n_+ be the space of $n \times n$ symmetric matrices and the cone of positive semidefinite matrices in S^n, respectively. Denote the Frobenius norm induced by the standard trace inner product $< \cdot, \cdot >$ in S^n by $\| \cdot \|$. Let C be a given matrix in S^n and $H \in S^n$ a given weight matrix whose entries are nonnegative. Then the rank constrained nearest correlation matrix problem (rank-NCM) can be formulated as follows:

$$\begin{aligned} & \min \quad \frac{1}{2}\|H \circ (X - C)\|^2 \\ \textbf{s.t.} \quad & X_{ii} = 1, \quad i = 1, \ldots, n, \\ & X \in S^n_+, \\ & rank(X) \leq r, \end{aligned}$$

where "$\circ$" denotes the Hadamard product, i.e., $(A \circ B)_{ij} = A_{ij}B_{ij},$, $i, j = 1, \ldots, n$ and $r \in \{1, \ldots, n\}$ is a given integer. The weight matrix H is introduced by adding larger weights to correlations that are better estimated or are of higher confidence in their correctness. Zero weights are usually assigned to those correlations that are missing or not estimated.

The weight matrix H is either the identity or the one provided by T.Fushiki at Institute of Statistical Mathematics, Japan.

According to T.Fushiki, if $\{(X_1, Y_1), \ldots, (X_N, Y_N)\}$ is assumed to be independent and identically distributed (i.i.d.), an asymptotic variance of the correlation coefficient between X and Y is obtained with the delta

method, which uses estimates of higher-order moments. If $\{(X_1, Y_1), \ldots, (X_N, Y_N)\}$ is, moreover, assumed to have a gaussian distribution, an asymptotic variance of $\widehat{\rho}_{X,Y}$ is obtained as $(1 - \widehat{\rho}^2_{X,Y})^2/N$ by using simply $\widehat{\rho}_{X,Y}$. Let $\widehat{V}_{ij}$ be an estimate of the variance of $\widehat{\rho}_{ij}$ and $\widehat{H}$ be the elementwise inverse of $\widehat{V}$:

$$\widehat{H}_{ij} = \begin{cases} 1/\widehat{V}_{ij} & \textbf{if } \widehat{V}_{ij} \neq 0 \\ 0 & \textbf{otherwise}. \end{cases}$$

If $\widehat{H}$ is used, the relaxed problem can be written as

$$\begin{aligned} \min \quad & \frac{1}{2}\mathbf{tr}[(X - \widehat{\rho})\{\widehat{H} \circ (X - \widehat{\rho})\}] \\ \textbf{s.t.} \quad & diag(X) = \mathbf{1}, \quad X \succeq 0. \end{aligned}$$

We express the rank constraints $rank(X) \leq r$ explicitly as $X = V^T V$ with $V = [V_1, \ldots, V_n] \in \mathbb{R}^{r \times n}$. Hence the original optimization problem is reduced to the minimization of a quadratic polynomial over spheres as follows:

$$\min_{V \in \mathbb{R}^{r \times n}} \quad \frac{1}{2}\|\widehat{H} \circ (V^T V - C)\|^2, \quad \textbf{s.t.} \quad \|V_i\|_2 = 1, i = 1, \ldots, n.$$

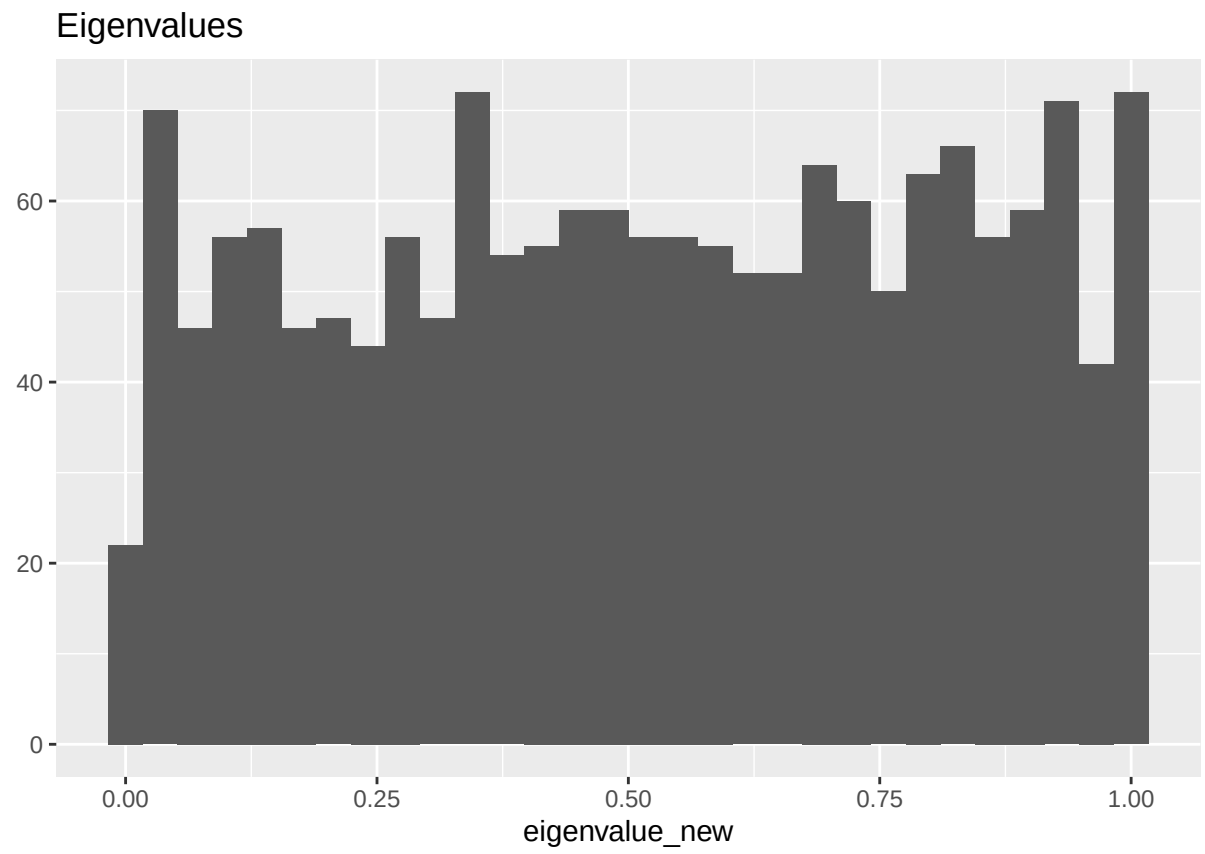
```
source('C:/Users/zhuzh/Desktop/Final/f and g.R')
source('C:/Users/zhuzh/Desktop/Final/OptM.R')
obj_sim <- as(sim, "matrix")
b <- matrix(rnorm(1664*1664), nrow = 1664, ncol = 1664)
b <- svd(b)$u
newb <- Beta(b,obj_sim)
newbb <- t(newb)%*%newb
norm(newbb)
```

```
## [1] 1
```

```
for (i in 31:1664){
  newbb[i,i] = 0+runif(1)
}

eigenvalue_new <- eigen(newbb)$values
eigenvalue_plot2 <- qplot(eigenvalue_new) + ggtitle('Eigenvalues')
eigenvalue_plot2
```

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



Model

We mainly implement 4 kinds of collaborative filtering systems. They are

- Item-based collaborative fltering system using cosine-based similarity;
- Item-based collaborative fltering system using correlation-based similarity;
- User-based collaborative fltering system using cosine-based similarity;
- User-based collaborative fltering system using correlation-based similarity.

User-based Collaborative Filtering Systems

User-based collaborative filtering systems are also called model-based collaborative fltering systems. Model-based collaborative fltering algorithms provide item recommendation by first developing a model of user ratings. Algorithms in this category take a probabilistic approach and envision the collaborative filtering process as computing the expected value of a user prediction, given his/her ratings on other items.

Item-based Collaborative Filtering Systems

Item-based collaborative filtering systems are also called memory-based collaborative fltering systems. Memory-based algorithms utilize the entire user-item database to generate a prediction. These systems employ statistical techniques to find a set of users, known as neighbors, that have a history of agreeing

with the target user (i.e., they either rate different items similarly or they tend to buy similar set of items). Once a neighborhood of users is formed, these systems use different algorithms to combine the preferences of neighbors to produce a prediction or top-N recommendation for the active user.

Cosine-based Similarity

In this case, two items are thought of as two vectors in the m dimensional user-space. The similarity between them is measured by computing the cosine of the angle between these two vectors. Formally, in the $m \times n$ ratings matrix, similarity between items i and j, denoted by $sim(i, j)$ is given by

$$sim(i, j) = cos(i, j) = \frac{i \cdot j}{\|i\|_2 * \|j\|_2}.$$

Correlation-based Similarity

In this case, similarity between two items i and j is measured by computing the $Pearson - r$ correlation $corr_{i,j}$. To make the correlation computation accurate we must first isolate the co-rated cases (i.e., cases where the users rated both i and j). Let the set of users who both rated i and j are denoted by U then the correlation similarity is given by

$$sim(i, j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_u)(R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_u)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_u)^2}}$$

Results

We first check the recommendation results of our simple model. Here are movies prediction for the first user.

```
recommend_n <- 10
model_try_predict <- predict(object = model_try,newdata=data_test,n=recommend_n)
user_1_try <- model_try_predict@items[[1]]
user_1_try_movie <- model_try_predict@itemLabels[user_1_try]
user_1_try_movie
```

```
##  [1] "Bed of Roses (1996)"              "Ice Storm, The (1997)"
##  [3] "In the Name of the Father (1993)" "How to Be a Player (1997)"
##  [5] "Deconstructing Harry (1997)"      "Spice World (1997)"
##  [7] "Deep Rising (1998)"               "Jeffrey (1995)"
##  [9] "Muriel's Wedding (1994)"          "Black Beauty (1994)"
```

We also check the results of the adjusted model with new correlation matrix. Notice that the results are different from before.

```
rownames(newbb) <- rownames(sim)
colnames(newbb) <- colnames(sim)

model_try_2 <- model_try
model_try_2@model$sim <- as(newbb,'dgCMatrix')

model_try_predict_2 <- predict(object = model_try_2,newdata=data_test,n=recommend_n)
user_1_try_2 <- model_try_predict_2@items[[1]]
user_1_try_movie_2 <- model_try_predict_2@itemLabels[user_1_try_2]
user_1_try_movie_2
```

```
##  [1] "Inventing the Abbotts (1997)" "Baton Rouge (1988)"
##  [3] "Bulletproof (1996)"           "Reality Bites (1994)"
##  [5] "Boys (1996)"                  "Last Man Standing (1996)"
##  [7] "Dante's Peak (1997)"          "Rosewood (1997)"
##  [9] "Kicked in the Head (1997)"    "In the Army Now (1994)"
```

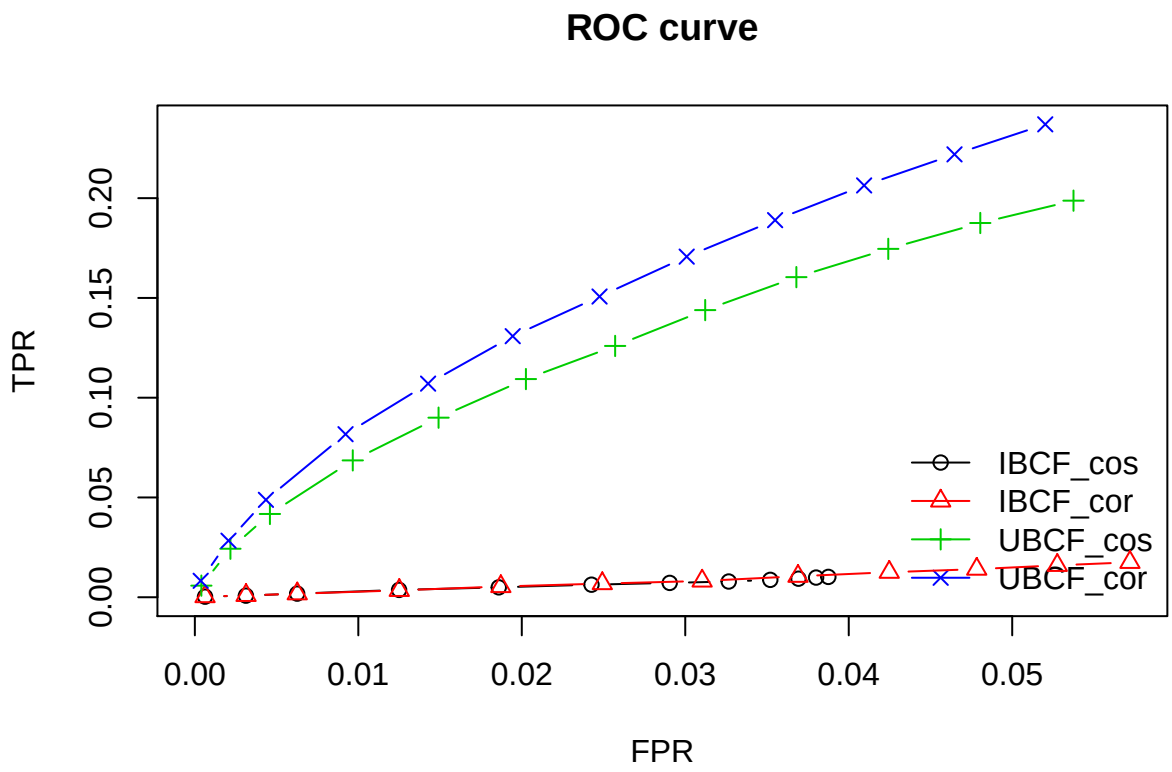
We then evalute the result of the four models mentioned above. From the ROC curve, we know that the UBCF algorithm based on pearson similarity performs best. The IBCF algorithms perform badly.

```
evalset <- evaluationScheme(data=MovieLense,method='cross-validation',k=4,given=15,goodRating=3)
models_to_evaluate <- list(
  IBCF_cos = list(name='IBCF',param=list(method='cosine')),
  IBCF_cor = list(name='IBCF',param=list(method='pearson')),
  UBCF_cos = list(name='UBCF',param=list(method='cosine')),
  UBCF_cor = list(name='UBCF',param=list(method='pearson'))
)

n_recommendations <- c(1, 5, seq(10, 100, 10))
list_results <- evaluate(x = evalset, method =
    models_to_evaluate, n = n_recommendations)
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [19.39sec/0.07sec]
##   2  [19.55sec/0.1sec]
##   3  [19.36sec/0.08sec]
##   4  [19.26sec/0.07sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [17.82sec/0.1sec]
##   2  [17.76sec/0.09sec]
##   3  [17.68sec/0.11sec]
##   4  [17.83sec/0.25sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0sec/1.45sec]
##   2  [0.02sec/1.42sec]
##   3  [0.02sec/1.41sec]
##   4  [0sec/1.4sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0sec/1.21sec]
##   2  [0sec/1.36sec]
##   3  [0sec/1.31sec]
##   4  [0.01sec/1.3sec]
```

```
plot(list_results)
title("ROC curve")
```



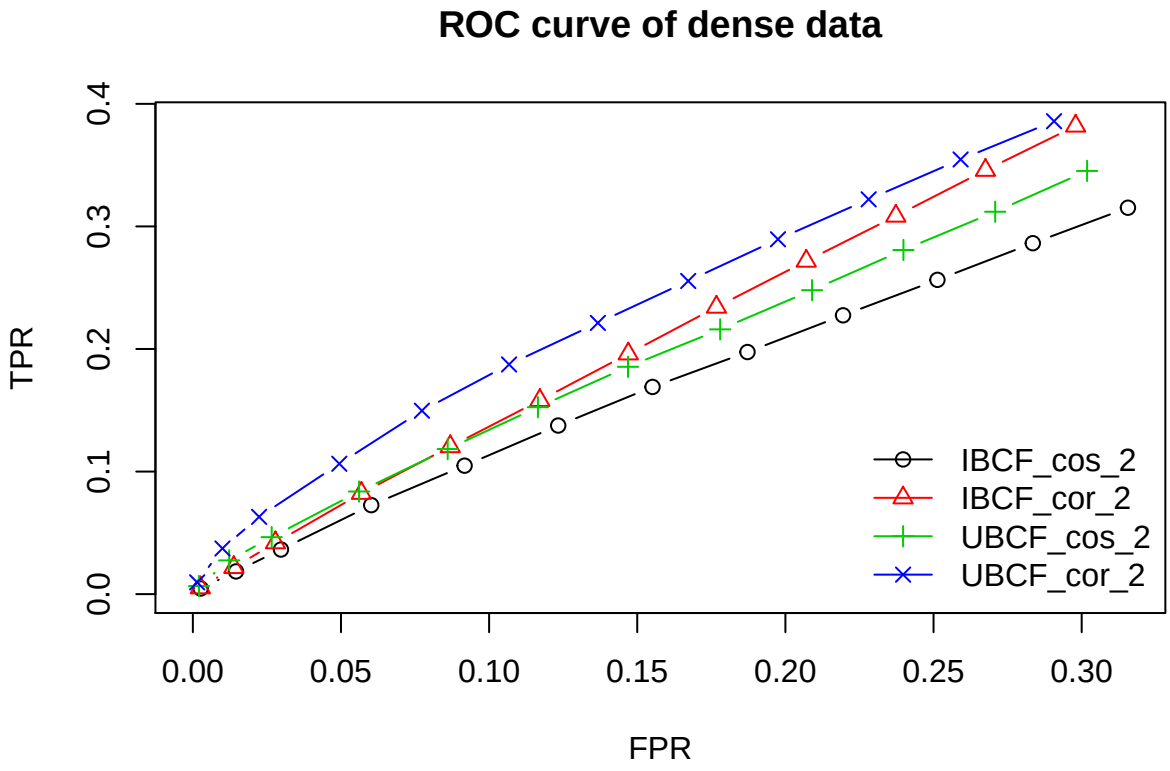
Notice that because of sparsity of the data, the IBCF algorithm has bad results. We rearrange the data to enhance the density.

```
densedata <- MovieLense[rowCounts(MovieLense) > 50,colCounts(MovieLense) > 100]
evalset_2 <- evaluationScheme(data=densedata,method='cross-validation',k=4,given=15,goodRating=3)
models_to_evaluate_2 <- list(
  IBCF_cos_2 = list(name='IBCF',param=list(method='cosine')),
  IBCF_cor_2 = list(name='IBCF',param=list(method='pearson')),
  UBCF_cos_2 = list(name='UBCF',param=list(method='cosine')),
  UBCF_cor_2 = list(name='UBCF',param=list(method='pearson'))
)
list_results_2 <- evaluate(x = evalset_2, method =
    models_to_evaluate_2, n = n_recommendations)
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.34sec/0.04sec]
##   2  [0.27sec/0.03sec]
##   3  [0.26sec/0.04sec]
##   4  [0.25sec/0.01sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.34sec/0.05sec]
##   2  [0.29sec/0.05sec]
##   3  [0.33sec/0.01sec]
##   4  [0.31sec/0.02sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0sec/0.17sec]
##   2  [0sec/0.2sec]
```

```
##   3  [0sec/0.2sec]
##   4  [0.01sec/0.17sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0sec/0.2sec]
##   2  [0.02sec/0.17sec]
##   3  [0sec/0.2sec]
##   4  [0sec/0.2sec]
```

```
plot(list_results_2)
title("ROC curve of dense data")
```



If the data is dense, all algorithms perform better than before. IBCF algorithms benefit a lot from dense data.