TP1 - Systèmes de Recommandation

Corentin Lacroix, Léa Boisdur

4 octobre 2016

0. Utility functions and data integration/preprocessing

```r
# imported packages
library(Matrix)

# Utility functions
# Cosinus entre un vecteur v et chaque colonne dela matrice m
cosinus.vm <- function(v,m) { n <- sqrt(colSums(m^2)); (v %*% m)/(n * sqrt(sum(v^2))) }

# Trouve les indexes des premières 'n' valeurs maximales d'une matrice
max.nindex <- function(m, n=5) {
  i <- order(m, decreasing=TRUE)
  return(i[1:n])
}

min.nindex <- function(m, n=5) {
    i <- order(m)
    return(i[1:n])
}

# Compute RMSR between two vector excluding NAs from computation
rmse <- function(a, b) {
  c <- a
  d <- b
  c[which(is.na(a))] = 0
  c[which(is.na(b))] = 0
  d[which(is.na(b))] = 0
  d[which(is.na(a))] = 0

    sqrt(mean((c - d)^2))
}

# Integrating data
u.user <- read.csv(file='./data/u.user.csv', sep='|', header=T)
u.item <- read.csv(file='./data/u.item.csv', sep='|', header=T)
u.data <- read.csv(file='./data/u.data.csv', sep='|', header=T)
```

1. Compute average rating by job and age

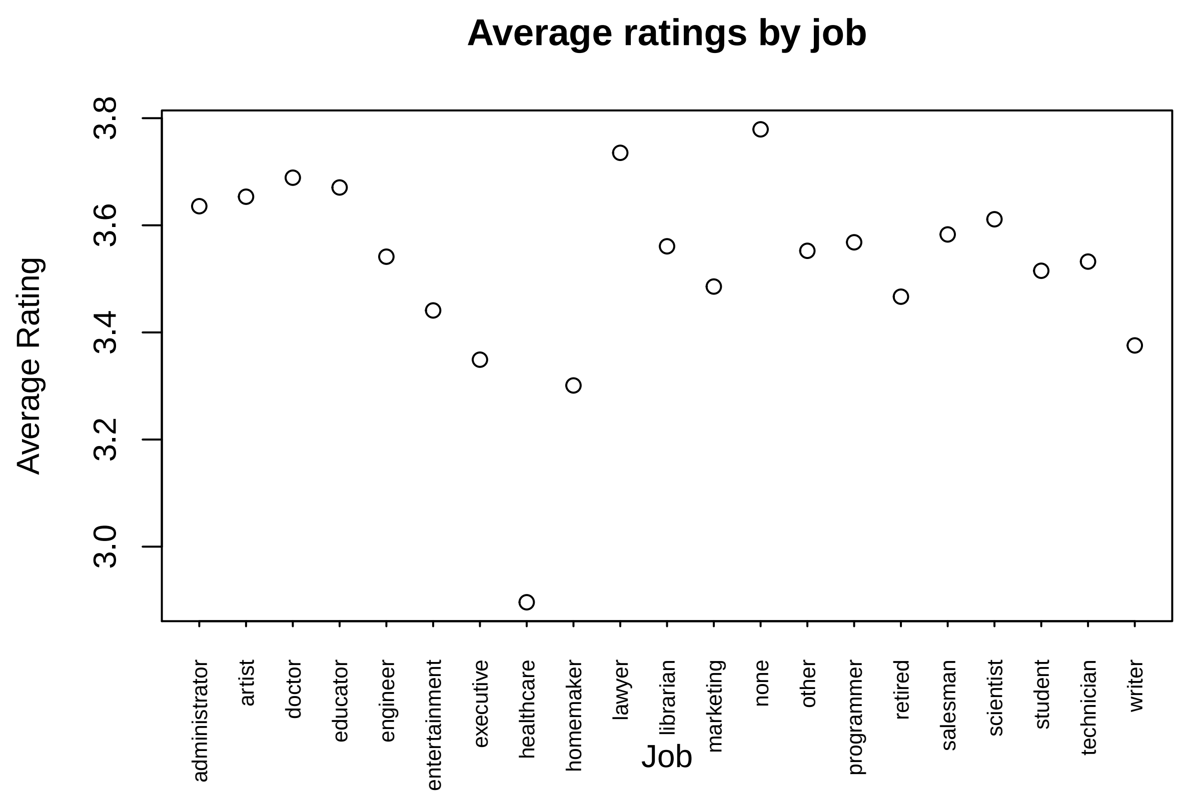
```r
# Change u.user columns names before join operations
colnames(u.user) <- c("user.id", "user.age", "user.gender", "user.job", "user.zip")

# Make dataframe combining u.data and u.user by Join operation on user.id column
fullData <- merge(x=u.data, y=u.user, by="user.id")
```

```r
# Job
jobMergedDf <- aggregate(x=fullData[,"rating"], by=list(job = fullData$user.job), FUN=mean)
print(jobMergedDf)
```

```
##              job        x
## 1  administrator 3.635646
## 2         artist 3.653380
## 3         doctor 3.688889
## 4       educator 3.670621
## 5       engineer 3.541407
## 6  entertainment 3.441050
## 7      executive 3.349104
## 8     healthcare 2.896220
## 9      homemaker 3.301003
## 10        lawyer 3.735316
## 11     librarian 3.560781
## 12     marketing 3.485641
## 13          none 3.779134
## 14         other 3.552377
## 15    programmer 3.568260
## 16       retired 3.466750
## 17      salesman 3.582944
## 18     scientist 3.611273
## 19       student 3.515143
## 20    technician 3.532230
## 21        writer 3.375723
```

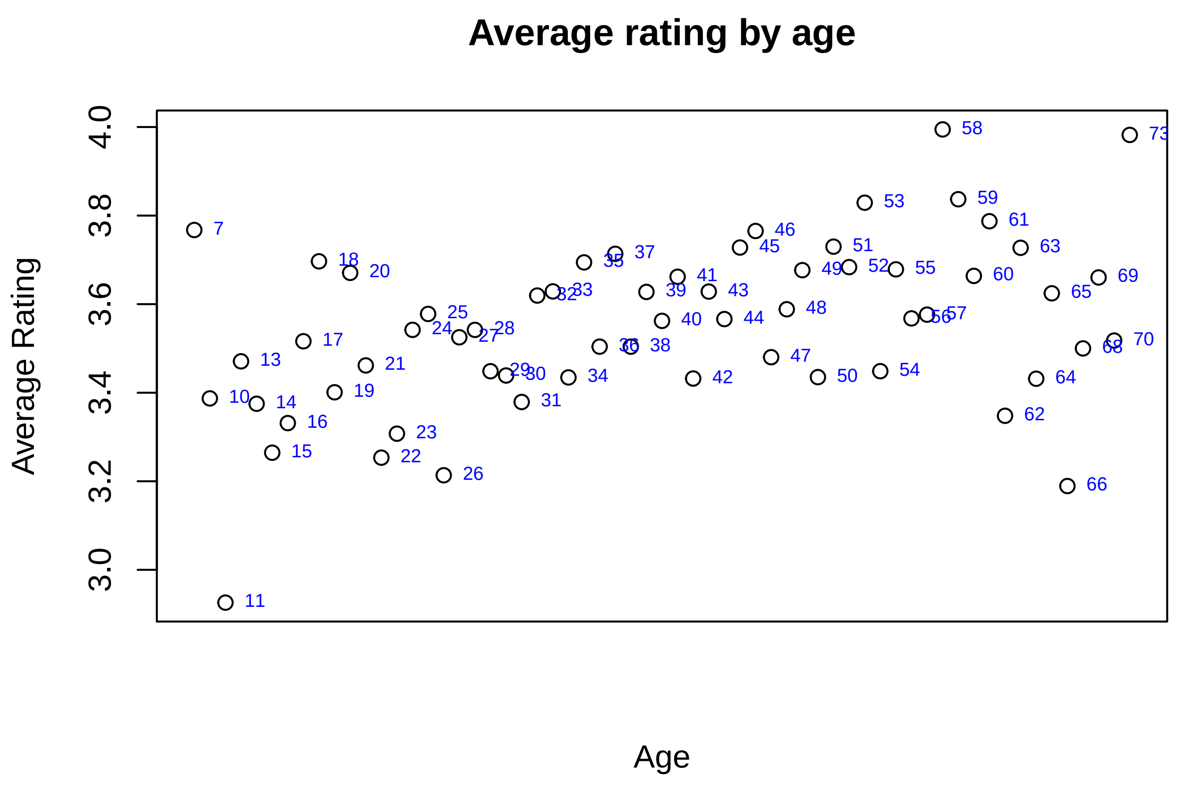
```r
plot(jobMergedDf$x, main="Average ratings by job", ylab="Average Rating", xlab="Job", xaxt="n")
axis(1, at=c(1:length(jobMergedDf$job)), labels=jobMergedDf$job, las=2, cex.axis=0.7, tck=-.01)
```



```r
# Age
ageMergedDf <- aggregate(x=fullData[,"rating"], by=list(age = fullData$user.age), FUN=mean)
print(ageMergedDf)
```

```
##    age        x
## 1    7 3.767442
## 2   10 3.387097
## 3   11 2.925926
## 4   13 3.470825
## 5   14 3.375000
## 6   15 3.264484
## 7   16 3.331343
## 8   17 3.516165
## 9   18 3.696710
## 10  19 3.400968
## 11  20 3.670580
## 12  21 3.461589
## 13  22 3.253330
## 14  23 3.307492
## 15  24 3.541923
## 16  25 3.577872
## 17  26 3.213531
## 18  27 3.525144
## 19  28 3.541862
## 20  29 3.448493
## 21  30 3.438862
## 22  31 3.379035
## 23  32 3.619399
## 24  33 3.628785
## 25  34 3.434505
## 26  35 3.694321
## 27  36 3.504039
## 28  37 3.713652
## 29  38 3.503983
## 30  39 3.627586
## 31  40 3.562194
## 32  41 3.661885
## 33  42 3.432019
## 34  43 3.628378
## 35  44 3.566276
## 36  45 3.727862
## 37  46 3.765201
## 38  47 3.480173
## 39  48 3.588406
## 40  49 3.676735
## 41  50 3.435140
## 42  51 3.729839
## 43  52 3.683625
## 44  53 3.829105
## 45  54 3.448598
## 46  55 3.678571
## 47  56 3.567797
## 48  57 3.576271
## 49  58 3.994550
## 50  59 3.836957
## 51  60 3.663766
## 52  61 3.787234
## 53  62 3.347826
## 54  63 3.727273
## 55  64 3.431579
## 56  65 3.624454
## 57  66 3.189189
## 58  68 3.500000
## 59  69 3.660256
## 60  70 3.517730
## 61  73 3.982143
```

```
plot(ageMergedDf$x, main="Average rating by age", ylab="Average Rating", xlab="Age", xaxt="n")
text(c(1:length(ageMergedDf$age)), ageMergedDf$x, ageMergedDf$age, cex=0.6, pos=4, col="blue")
```



2. Compute 10 more similar movies to Star Trek using cosinus distance coefficients and correlation coefficients between movies

```
# Make User-Item Matrix : a sparse matrix containing 0s and a dense matrix containing NAs
UIM.sparse <- sparseMatrix(i = u.data$user.id,
                           j = u.data$item.id,
                           x = u.data$rating)
rownames(UIM.sparse) <- paste('u', 1:nrow(UIM.sparse), sep='')
colnames(UIM.sparse) <- paste('i', 1:ncol(UIM.sparse), sep='')

# Make also a dense matrix from this UI sparse matrix
UIM <- as.matrix(UIM.sparse)
UIM[UIM ==0] <- NA

# Construct vector with cosinus distance between interest movie and all others
cosItem <- as.vector(cosinus.vm(UIM.sparse[,u.item[u.item$movie.title == "Star Trek V: The Final Fronti
corItem <- as.vector(cor(x=UIM.sparse[,450], y=as.matrix(UIM.sparse)))

# Take the 10 closest movies (removing itself in fisrt position of cosItem and corItem)
# Cosinus distance
u.item[u.item$movie.id %in% tail(max.nindex(cosItem,11),10),"movie.title"]
```

```
##  [1] Stargate (1994)
##  [2] Die Hard 2 (1990)
##  [3] Star Trek VI: The Undiscovered Country (1991)
##  [4] Star Trek: The Wrath of Khan (1982)
##  [5] Star Trek III: The Search for Spock (1984)
##  [6] Star Trek IV: The Voyage Home (1986)
##  [7] Star Trek: Generations (1994)
##  [8] Star Trek: The Motion Picture (1979)
##  [9] Escape from New York (1981)
## [10] Conan the Barbarian (1981)
## 1664 Levels: 101 Dalmatians (1996) 12 Angry Men (1957) ... Zeus and Roxanne (1997)
```

```r
# Correlation coefficient "distance"
u.item[u.item$movie.id %in% tail(max.nindex(corItem,11),10),"movie.title"]
```

```
##  [1] Stargate (1994)
##  [2] Die Hard 2 (1990)
##  [3] Star Trek VI: The Undiscovered Country (1991)
##  [4] Star Trek: The Wrath of Khan (1982)
##  [5] Star Trek III: The Search for Spock (1984)
##  [6] Star Trek IV: The Voyage Home (1986)
##  [7] Star Trek: Generations (1994)
##  [8] Star Trek: The Motion Picture (1979)
##  [9] Escape from New York (1981)
## [10] Conan the Barbarian (1981)
## 1664 Levels: 101 Dalmatians (1996) 12 Angry Men (1957) ... Zeus and Roxanne (1997)
```

3. Use an item-item approach to predict ratings for users who haven't rated Star Trek

a. Compute vectors containing Euclidean Distance between Star Trek and all ohters movies and compute sorted (by index number) list of 21 NN

```r
n.voisins = 21

# With no votes being 0s in sparse UI matrix
distance.450 <- sqrt(colSums((UIM.sparse[,450] - UIM.sparse)^2))
i.distance.450.NN <- sort(min.nindex(distance.450, n.voisins))

# With no votes being NAs in dense UI Matrix
distance.na.450 <- sqrt(colSums((UIM[,450] - UIM)^2, na.rm=T))
i.distance.na.450.NN <- sort(min.nindex(distance.na.450, n.voisins))

# With distances being computed by R dist() function on sparse UI Matrix
#distance.dist.na.450 <- as.matrix(dist(t(UIM)))[450,]
#i.distance.dist.na.450.NN <- sort(min.nindex(distance.dist.na.450, n.voisins))

#with distances being computed by R dist() function on dense UI Matrix
#distance.dist.450 <- as.matrix(dist(t(UIM.sparse)))[450,]
#i.distance.dist.450.NN <- sort(min.nindex(distance.dist.450, n.voisins))
```

All these distances were computed for understanding purpose only. Only one of them (i.distance.450) was used for the rest of the question.

b. Compute vectors of cosinus distance between Star Trek and its 20 NN (using i.distance.450 as the NN vector)

```r
# Compute cosinus coeffcients between Star Trek and its 20 Nearest Neighbors
cos.450.21N <- as.vector(cosinus.vm(UIM.sparse[,450], UIM.sparse[,i.distance.450.NN]))

# Extract the full votes vector for Star Trek
votes.450 <- UIM[,450]

# Make temporary matrix by filtering in UIM.sparse lines corresponding to users who haven't voted for S
tmp.mat <- as.matrix(UIM.sparse[which(is.na(votes.450)),i.distance.450.NN])

# Compute vector of cosinus weights sum for final predicted votes (cosinus coefficient for neighbors it
weights.sum <- ifelse(tmp.mat == 0,0,1) %*% cos.450.21N

# Replace missing votes in votes.450 by predicted vote (as stated before the user must have rated at le
predictedVotes.450 <- votes.450
predictedVotes.450[which(is.na(votes.450))][weights.sum > 0] <- as.vector(tmp.mat[weights.sum > 0,] %*%
```

4. Compute quadratic error of the previous item-item approach

```r
# We take the real 20 NN neighbors of Star Trek (excluding himself)
i.distance.450.NN <- tail(i.distance.450.NN, 20)

# We get a slice of the sparse UI Matrix containing only votes for the 20NN of ST
tmp.mat <- as.matrix(UIM.sparse[,i.distance.450.NN])

# We compute the vector of cosinus weights between ST and the other movies
cos.450.20N <- as.vector(cosinus.vm(UIM.sparse[,450], UIM.sparse[,i.distance.450.NN]))

# We compute for all users the sum of cosinus weights corresponding to movies that the users have rated
weights.sum <- ifelse(tmp.mat == 0,0,1) %*% cos.450.20N

# We compute the predicted votes for ST for users that have rated at least one of the Star Trek 20NN (i
predictedVotes.450 <- rep(0, nrow(UIM.sparse))
predictedVotes.450[weights.sum > 0] <- as.vector(tmp.mat[weights.sum > 0,] %*% cos.450.20N) / weights.s

# We compute the RMSE of the prediction process for all users for which we predicted a vote and that ha
rmseScore <- rmse(predictedVotes.450, votes.450)
rmseScore
```

```
## [1] 0.5698663
```

We can note that this method for estimating the RMSE of the prediction process is biased : we predict votes that have already been used to compute the 20 nearest neighbors items of Star Trek. This score is consequently too optimist.

To avoid this we should split the data into "train/test" like datasets : one containing ratings of 2/3 of the Database users, one containing the last 1/3 of the users database. We then could compute NN of Star Trek based on ratings from "train users", predict ratings for "test users" and measure RMSE on these predicted votes. We could also repeat this process k times to get a more exact estimation of the prediction method

RMSE score (k-fold cross validation). We thought that this was a little out of the scope of this assignment and consequently didn't implement it.

5. Compute 10 best movies suggestions for a manually created user using a user-user and NN approach

```r
# Construct a vector containing indexes of target movies
indices.starTrek <- grep("trek", as.character(u.item$movie.title), ignore.case=T)
indices.starWars <- c(172,181)
indices <- c(indices.starTrek, indices.starWars)

# Construct the vote vector for the user u
user.votes <- rep(0, ncol(UIM.sparse))
user.votes[indices.starTrek] = 5
user.votes[indices.starWars] = 1

# Compute vector containing common votes with database users
user.votes.communs <- ((UIM.sparse > 0) + 0) %*% ((user.votes > 0) + 0) # nombre de votes communs

# Compute distances between the constructed user and Database users and compute its 20 NN : for predict
distance.user <- sqrt(colSums((user.votes - t(UIM.sparse))^2))
i.distance.user.NN <- names(sort(distance.user[which(user.votes.communs >= 3)])[1:20])

# Compute inter-users cosinus coefficients (between the constructed User and its 20NN only)
cos.user.20NN <- as.vector(cosinus.vm(user.votes, t(UIM.sparse[i.distance.user.NN,])))

# Compute vector of cosinus weights sum between user and database users
tmp.mat <- as.matrix(UIM.sparse[i.distance.user.NN, ])
weights.sum <- ifelse(t(tmp.mat) == 0,0,1) %*% cos.user.20NN

# Make predictions for movies rated by at least 1 NN of the constructed User (weights.sum > 0) and not
user.predictedVotes <- rep(0, ncol(UIM.sparse))
user.predictedVotes[-c(which(weights.sum == 0), indices)] <- t(tmp.mat)[-c(which(weights.sum == 0), indi

# Get titles of the 10 best rated movies
u.item[u.item$movie.id %in% max.nindex(user.predictedVotes, 10), "movie.title"]
```

```
##  [1] Richard III (1995)
##  [2] Lone Star (1996)
##  [3] Spitfire Grill, The (1996)
##  [4] Bound (1996)
##  [5] 12 Angry Men (1957)
##  [6] Black Sheep (1996)
##  [7] Man Without a Face, The (1993)
##  [8] Day the Earth Stood Still, The (1951)
##  [9] Duck Soup (1933)
## [10] East of Eden (1955)
## 1664 Levels: 101 Dalmatians (1996) 12 Angry Men (1957) ... Zeus and Roxanne (1997)
```

We had some difficulties for this question : in the first steps of our implementation, suggestions were very

very poor. We then tried to exclude all users that have not at least 3 common ratings with the user from its potential neighbors and suggestions became more relevant (but not much).

Predictions computed by this method are quite surprising and hard to understand. We haven't deeply investigated why these items are suggested but we have obsersed several things about these recommendations :

- items from the beginning of the items database are more likely to be rated by this method (this is in fact related to the fact that items at the beginning of the items DataBase are more famous in a way than the ones at the end)
- there is not any strong links between movies liked by the constructed User (Star Trek) and the suggested movies
- None of the movies close to Star Trek have been recommended (ex : Star Gate)
- our guess is that all of these suggested movies have good average ratings

We have several explanations for these results :

- Suggestions don't seem very good because using Collaborative filtering User-User approach is not a good approach in this case : UI matrix is very sparse, especially in the user dimension. The probability that 2 users have common ratings is much lower than probability that 2 items have common ratings. This tends to make the predictions less revelant.
- None of the closest Star Trek movies can be recommended by this method because they probably also are very close to Star Wars movies which were given 1/5 ratings... Our guess is that close users to our User gave also poor ratings or medium ratings to Star Wars movies.
- As any clear rating pattern is learned by the method, suggested movies are likely to be "Good average ratings" movies (like 12 Angry Men or East of Eden as far as we know)

6. Make 10 movies suggestions for new users based on users information (age, gender, job)

For this question, we have decided to use Naïve Bayes method for predicting user votes for movies, and then taking the 10 movies with higher predicted rating. We explored 2 strategies for dealing with the problem :

Strategy 1 : for each movie, we compute a Naïve Bayesian Model based on the ratings for the movie. Advantages :

- It is the more natural method for this problem.

Drawbacks :

- It is likely that for many movies, there will be only a small number of observations, making the predictions irrelevant.
- Moreover, for movies without ratings from users sharing common characteristics with the target user, we won't be able to make any predictions on the rating. In this model, $P(R = k|A = a, G = g, J = j)$ is linked to $P(R = k)P(R = k|A = a)P(R = k|G = g)P(R = k|J = j)$. So for example, if any writer has never rated the movie Star Trek, it won't possible to estimate $P(R = k|J = writer)$, neither $P(R = k|A = a, G = g, J = writer)$.
- There is another problem of the same kind. For example, as soon as any writer hasn't given a 3 rating for Star Trek, the quantity $P(R = 3)P(R = 3|A = a)P(R = 3|G = g)xP(R = 3|J = j)$ is impossible to estimate (or 0). This will also affect others probabilities $P(P(R = k|A = a, G = g, J = j)$ where $k \neq 3$ (this will tend to increase them). The Laplace approximation should help reducing this problem.

Strategy 2 : for each kind of movie (1 kind of movie = 1 or more combination of the binary variables set describing the movie in the Dataset) Advantages :

- There will be less models to compute since there are much less movie types than movies. Moreover, there should be more observations (= ratings + content information on user) to compute the models.

Drawbacks :

- With this method, we can only suggest a few different types movies and in most case, we will suggest 10 movies of the same type. Moreover, all movies of the same type will have the same predicted rating and we would have to combine this model with others that allow us to rank movies of the same type.
- The same kind of problems than for the 1st strategy could still be observed.

For both strategies : If we consider the age as a categorical variable with as many levels as different ages exist, this will lead to very poor results : the algorithm will produce very poor and irrelevant approximations of probabilities P(R=k|Age=a) because for many movies only a few users with the same age than our target user have rated the movie.

In the Naïve Bayes framework, We could consider the variable age as a Gaussian variable. We could also transform this variable into a categorical variable with less categories. This is what we have done for this question.

We have chosen to implement an algorithm based on the 1st strategy that seems the more relevant for this problem. We have used the Naïve Bayes implementation from the R-cran package e1071.

```r
library(e1071)

# Make age categorical (8 different age classes)
user.age.hist <- hist(u.user$user.age, breaks = c(0,16,20,25,30,37,45,55,100), freq=FALSE, plot=FALSE)
```

```
## Warning in hist.default(u.user$user.age, breaks = c(0, 16, 20, 25, 30,
## 37, : argument 'freq' is not made use of
```

```r
u.user.bis <- u.user
u.user.bis$user.age <- cut(u.user$user.age, breaks = user.age.hist$breaks, labels = c(1:8))

# Make global dataframe for computing models
nb_df <- merge(x=u.data, y=u.user.bis, by="user.id")
nb_df <- nb_df[,!names(nb_df) %in% c("user.zip", "timestamp")]
nb_df <- as.data.frame(lapply(nb_df, factor))

# Function that returns predict votes for all movies given the characteristics of the input user. 0 vot

predictRatings <- function(user_char, data, nmovies) {
  ratings = rep(0, nmovies)

  # We try to predict ratings for all movies
  for (movie in c(1:nmovies)) {
    movie_filter <- data$item.id == movie
    filtered_data <- data[movie_filter,]
    nobs <- nrow(data[movie_filter,])
    # If there is not enough observations, we chose to not make predictions for the movies
    if (nobs > 5) {
      # The following integers are the number of observations (= ratings) whose user share the same age
      n1 <- nrow(filtered_data[filtered_data$user.age == user_char[1],])
      n2 <- nrow(filtered_data[filtered_data$user.gender == user_char[2],])
      n3 <- nrow(filtered_data[filtered_data$user.job == user_char[3],])

      if (n1 * n2 * n3 > 0) {
        # We build the model with alpha in Laplace approximation being 1
        m <- naiveBayes(rating ~ user.age + user.gender + user.job, data = filtered_data, laplace = 1)

        # We have to set NaN values in conditional tables of the models by 0s, otherwise the rating com
        m$tables$user.age[is.na(m$tables$user.age)] = 0
        m$tables$user.gender[is.na(m$tables$user.gender)] = 0
        m$tables$user.job[is.na(m$tables$user.job)] = 0

        # We make a single row Dataframe for our target user
        user <- data.frame(user.id = NA, item.id = NA, rating = NA, user.age = factor(user_char[1], leve

        # The predicted rating is not the most probable rating but the probability-weighted average of
        rating <- sum(predict(m, user, type="raw") * c(1:5), na.rm = TRUE)
        ratings[movie] = rating
        }
    }
  }
  return(ratings)
}

# Test code
nmovies = length(levels(nb_df$item.id))
user.char = c(3,"F","healthcare")
predictedVotes <- predictRatings(user.char, nb_df, nmovies)

# If some predictions have been made, print 10 best suggestions otherwise print an error message
if (sum(predictedVotes == rep(0, nmovies)) > 0) {
  u.item[u.item$movie.id %in% max.nindex(predictedVotes, 10), "movie.title"]
} else {
    print("Error : no prediction could be made for this features combination")
}
```

```
##  [1] Fugitive, The (1993)
##  [2] Kolya (1996)
##  [3] Titanic (1997)
##  [4] Schindler's List (1993)
##  [5] Paradise Lost: The Child Murders at Robin Hood Hills (1996)
##  [6] Day the Earth Stood Still, The (1951)
##  [7] Shadowlands (1993)
##  [8] Ghost and Mrs. Muir, The (1947)
##  [9] Safe (1995)
## [10] Mina Tannenbaum (1994)
## 1664 Levels: 101 Dalmatians (1996) 12 Angry Men (1957) ... Zeus and Roxanne (1997)
```

This method depends on a few parameters (minimum number of observations for computing predictions, alpha of the Laplace approximation) that should be optimized using a score metric and Cross-Validation. We tested our method with few different users. With this configuration, movies suggestions seems to be relevant even if suggested movies tend to be movies with very high average rating.