HarvardX - PH125.9x |Data Science: Capstone

MovieLens Project

Julien Geffroy

11.02.2022

A. Create train and validation sets

```
##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(data.table)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
          col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

# if using R 3.6 or earlier:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                   title = as.character(title),
                                   genres = as.character(genres))
# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                   title = as.character(title),
                                   genres = as.character(genres))
```

```
movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
    semi_join(edx, by = "movieId") %>%
    semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

B. Data Exploration:

⇨ *see Quiz: MovieLens Dataset*

C. Data transformation

```
#Edx dataset transformation:usersId and movieId should be treat as factors for some analysis purposes.
edx.copy <- edx
edx.copy$userId <- as.factor(edx.copy$userId)
edx.copy$movieId <- as.factor(edx.copy$movieId)
```

```
#SparseMatrix function is used in order to get an output 0f sparse matrix of class dgcMatrix.
# To use this function, the userId & movieId are converted to numeric vectors.
edx.copy$userId <- as.numeric(edx.copy$userId)
edx.copy$movieId <- as.numeric(edx.copy$movieId)
sparse_ratings <- sparseMatrix(i = edx.copy$userId,
j = edx.copy$movieId ,
x = edx.copy$rating,
dims = c(length(unique(edx.copy$userId)),
length(unique(edx.copy$movieId))),
dimnames = list(paste("u", 1:length(unique(edx.copy$userId)), sep = ""),
paste("m", 1:length(unique(edx.copy$movieId)), sep = "")))
```

```
#Remove the copy created
rm(edx.copy)
```

```
#The first 10 users
sparse_ratings[1:10,1:10]
```

```
## 10 x 10 sparse Matrix of class "dgCMatrix"
## [[ suppressing 10 column names 'm1', 'm2', 'm3' ... ]]
##
## u1 . . . . . . . . . .
## u2 . . . . . . . . . .
## u3 . . . . . . . . . .
## u4 . . . . . . . . . .
## u5 1 . . . . . 3 . . .
## u6 . . . . . . . . . .
## u7 . . . . . . . . . .
## u8 . 2.5 . . 3 4 . . . .
## u9 . . . . . . . . . .
## u10 . . . . . . 3 . . .
```

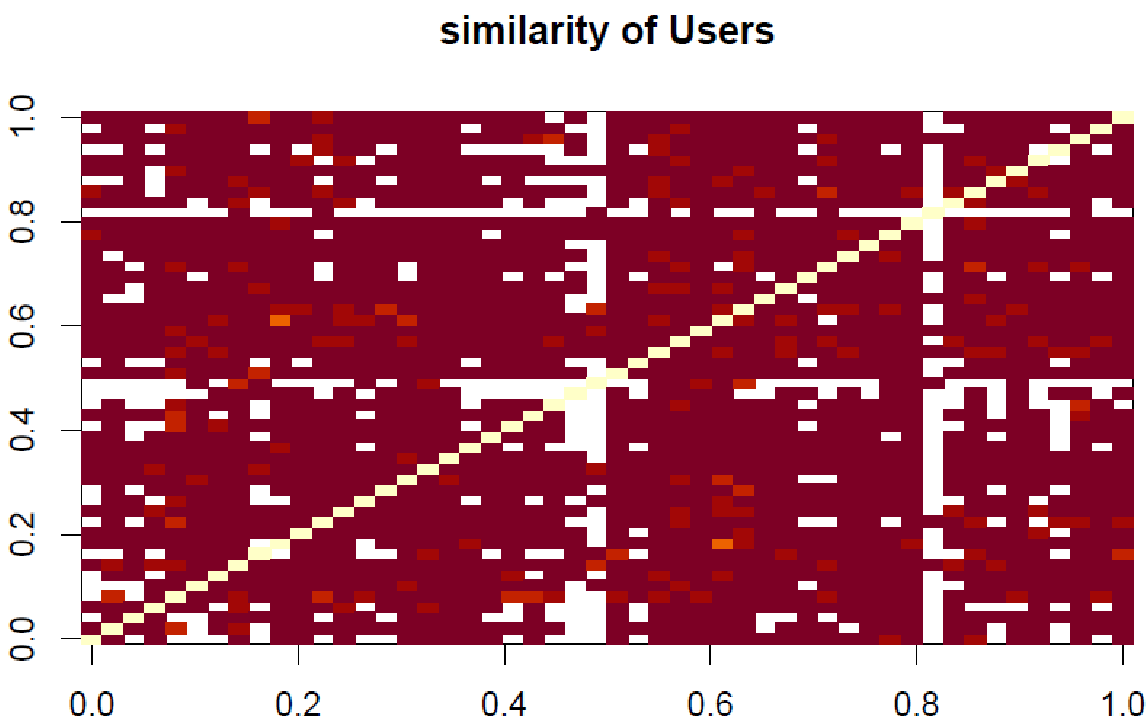
```r
##Convert rating matrix into a recommenderlab sparse matrix
rate_Mat <- new("realRatingMatrix", data = sparse_ratings)
rate_Mat
```

```
## 69878 x 10677 rating matrix of class 'realRatingMatrix' with 9000055 ratings.
```

D. Similarity measures

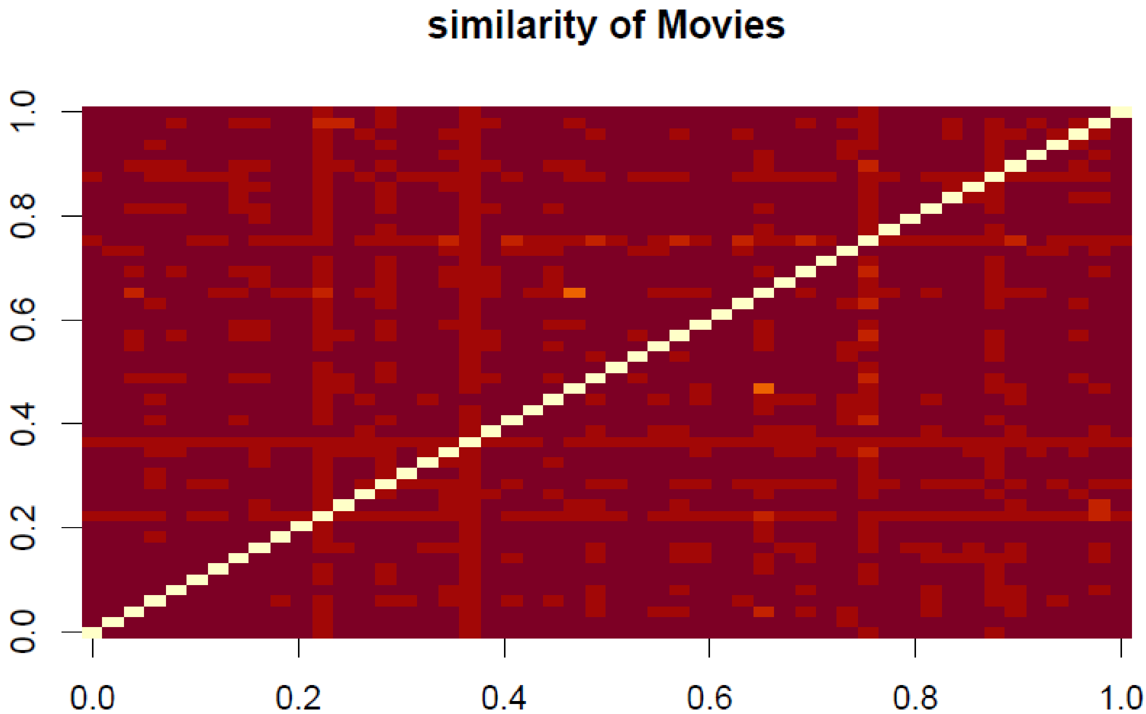
```r
#calculate the user similarity using the cosine similarity
similarity_users <- similarity(rate_Mat[1:50,],
method = "cosine",
which = "users")
```

```r
image(as.matrix(similarity_users), main = "similarity of Users")
```




```
#Using the same approach, compute similarity between movies.
similarity_movies <- similarity(rate_Mat[,1:50],
method = "cosine",
which = "items")
```

```
image(as.matrix(similarity_movies), main = "similarity of Movies")
```



E. Dimension reduction

```r
#implicitly restarted Lanczos bidiagonalization algorithm (IRLBA)
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

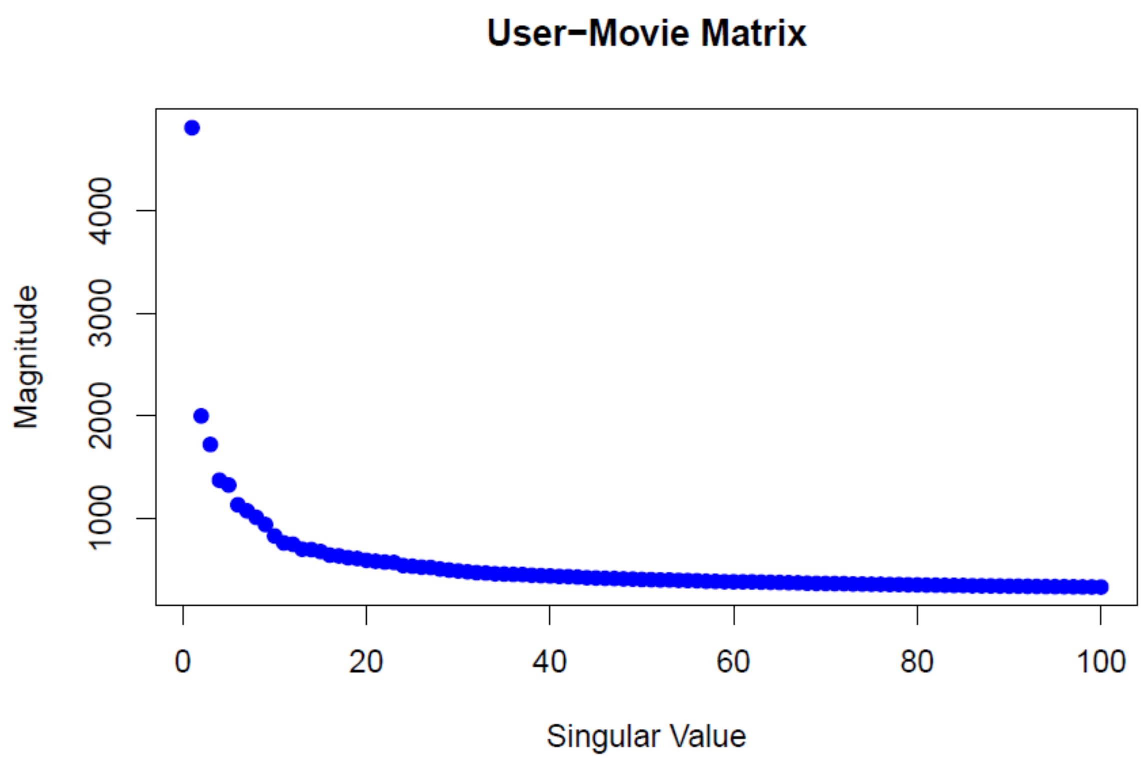
```r
Y <- irlba(sparse_ratings,tol=1e-4,verbose=TRUE,nv = 100, maxit = 1000)
```

```
## Working dimension size 107

## Initializing starting vector v with samples from standard normal distribution.
## Use `set.seed` first for reproducibility.

## irlba: using fast C implementation
```

```r
# plot singular values
plot(Y$d, pch=20, col = "blue", cex = 1.5, xlab='Singular Value', ylab='Magnitude',
main = "User-Movie Matrix")
```




```
# calculate sum of squares of all singular values
all_sing_val <- sum(Y$d^2)
```

```
# variability described by first 6, 12, and 20 singular values
first_six <- sum(Y$d[1:6]^2)
print(first_six/all_sing_val)
```

```
## [1] 0.6187623
```

```
first_twl <- sum(Y$d[1:12]^2)
print(first_twl/all_sing_val)
```

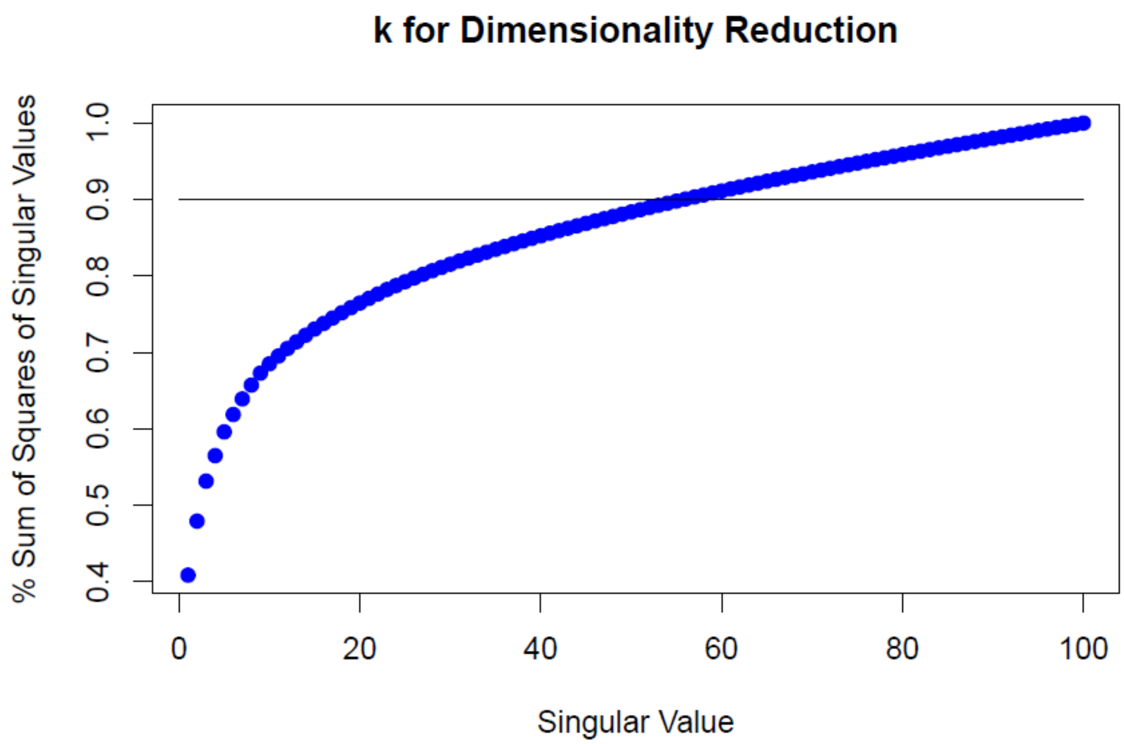
```
## [1] 0.7052297
```

```
first_twt <- sum(Y$d[1:20]^2)
print(first_twt/all_sing_val)
```

```
## [1] 0.7646435
```

```
perc_vec <- NULL
for (i in 1:length(Y$d)) {
perc_vec[i] <- sum(Y$d[1:i]^2) / all_sing_val
}
```

```r
plot(perc_vec, pch=20, col = "blue", cex = 1.5, xlab='Singular Value', ylab='% Sum of Squares of Singular
lines(x = c(0,100), y = c(.90, .90))
```




```r
#value of K
k = length(perc_vec[perc_vec <= .90])
k
```

[1] 55

```r
#get the decomposition of Y ; matrices U, D, and V
U_k <- Y$u[, 1:k]
dim(U_k)
```

[1] 69878 55

```r
D_k <- Diagonal(x = Y$d[1:k])
dim(D_k)
```

[1] 55 55

```r
V_k <- t(Y$v)[1:k, ]
dim(V_k)
```

[1] 55 10677

F. Relevant Data

```
#1. Determine the minimum number of movies per user.
min_no_movies <- quantile(rowCounts(rate_Mat), 0.9)
print(min_no_movies)
```

```
## 90%
## 301
```

```
#2. Determine the minimum number of users per movie.
min_no_users <- quantile(colCounts(rate_Mat), 0.9)
print(min_no_users)
```

```
## 90%
## 2150.2
```

```
#3. Select the users and movies matching these criteria.
rate_movies <- rate_Mat[rowCounts(rate_Mat) > min_no_movies,
colCounts(rate_Mat) > min_no_users]
rate_movies
```

```
## 6978 x 1068 rating matrix of class 'realRatingMatrix' with 2313148 ratings.
```

Movie effect

```
#calculate the average of all ratings of the edx dataset
mu <- mean(edx$rating)
```

```
#calculate b_i on the training dataset
movie_m <- edx %>%
group_by(movieId) %>%
summarize(b_i = mean(rating - mu))
```

```
# predicted ratings
predicted_ratings_bi <- mu + validation %>%
left_join(movie_m, by='movieId') %>%
.$b_i
```

Movie and user effect

```
#calculate b_u using the training set
user_m <- edx %>%
left_join(movie_m, by='movieId') %>%
group_by(userId) %>%
summarize(b_u = mean(rating - mu - b_i))
```

```
#predicted ratings
```

```
predicted_ratings_bu <- validation %>%
left_join(movie_m, by='movieId') %>%
left_join(user_m, by='userId') %>%
mutate(pred = mu + b_i + b_u) %>%
.$pred
```

Movie, user and time effect

```
#create a copy of validation set , valid, and create the date feature which is the timestamp converted to
valid <- validation
valid <- valid %>%
mutate(date = round_date(as_datetime(timestamp), unit = "week"))
```

```
#calculate time effects ( b_t) using the training set
temp_m <- edx %>%
left_join(movie_m, by='movieId') %>%
left_join(user_m, by='userId') %>%
mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
group_by(date) %>%
summarize(b_t = mean(rating - mu - b_i - b_u))
```

```
#predicted ratings
predicted_ratings_bt <- valid %>%
left_join(movie_m, by='movieId') %>%
left_join(user_m, by='userId') %>%
left_join(temp_m, by='date') %>%
mutate(pred = mu + b_i + b_u + b_t) %>%
.$pred
```

The root mean square error (RMSE) models for movies, users and time effects

```
#calculate the RMSE for movies
rmse_model_1 <- RMSE(validation$rating,predicted_ratings_bi)
rmse_model_1
```

```
## [1] 0.9439087
```

```
#calculate the RMSE for users
rmse_model_2 <- RMSE(validation$rating,predicted_ratings_bu)
rmse_model_2
```

```
## [1] 0.8653488
```

```
#calculate the RMSE for time effects
rmse_model_3 <- RMSE(valid$rating,predicted_ratings_bt)
rmse_model_3
```

```
## [1] 0.8652511
```

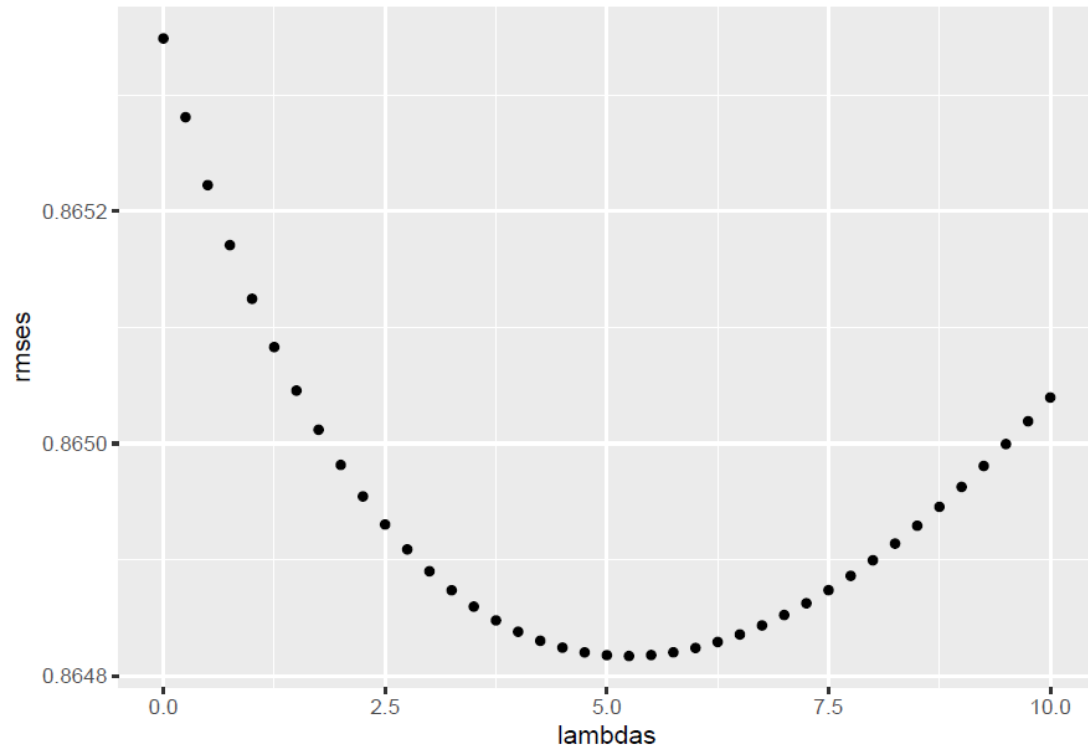
From the movie and user effects combined, our RMSE decreased by almost 10% with respect to the only movie effect. The improvement on the time effect is not significant, (about a decrease of 0.011%). The regularization would be performed using only the movie and user effects.

```
#remove valid before regularization
rm(valid)
```

G. Regularization

```
## remembering that lambda is a tuning parameter. We can use cross-validation to choose it
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
mu_reg <- mean(edx$rating)
b_i_reg <- edx %>%
group_by(movieId) %>%
summarize(b_i_reg = sum(rating - mu_reg)/(n()+l))
b_u_reg <- edx %>%
left_join(b_i_reg, by="movieId") %>%
group_by(userId) %>%
summarize(b_u_reg = sum(rating - b_i_reg - mu_reg)/(n()+l))
predicted_ratings_b_i_u <-
validation %>%
left_join(b_i_reg, by = "movieId") %>%
left_join(b_u_reg, by = "userId") %>%
mutate(pred = mu_reg + b_i_reg + b_u_reg) %>%
.$pred
return(RMSE(validation$rating,predicted_ratings_b_i_u))
})
```

```
qplot(lambdas, rmses)
```



The optimal lambda for the full model

```
#For the full model, the optimal Î» is given as
lambda <- lambdas[which.min(rmses)]
lambda
```

[1] 5.25

```
rmse_model_4 <- min(rmses)
rmse_model_4
```

[1] 0.864817

Summary of the rmse on validation set for Linear regression models

```
#summarize all the rmse on validation set for Linear regression models
rmse_results <- data.frame(methods=c("movie effect","movie + user effects","movie + user + time
effects",

kable(rmse_results) %>%
kable_styling(bootstrap_options = "striped" , full_width = F , position = "center") %>%
kable_styling(bootstrap_options = "bordered", full_width = F , position ="center") %>%
column_spec(1,bold = T ) %>%
column_spec(2,bold =T ,color = "white" , background ="#D7261E")
```

methods	rmse
movie effect	0.9439087
movie + user effects	0.8653488
movie + user + time effects	0.8652511
Regularized Movie + User Effect Model	0.8648170

The regularization gets down the RMSE's value to 0.8648170.
POPULAR , UBCF and IBCF algorithms of the recommenderlab package

```
#POPULAR algorithms of the recommenderlab package
model_pop <- Recommender(rate_movies, method = "POPULAR",
param=list(normalize = "center"))
```

```
#prediction example on the first 10 users
pred_pop <- predict(model_pop, rate_movies[1:10], type="ratings")
as(pred_pop, "matrix")[,1:10]
```

Rmse for popularity based recommender engine

```
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
#Calculation of rmse for popular method
eval <- evaluationScheme(rate_movies, method="split", train=0.7, given=-5)
```

```
#ratings of 30% of users are excluded for testing
model_pop <- Recommender(getData(eval, "train"), "POPULAR")

prediction_pop <- predict(model_pop, getData(eval, "known"), type="ratings")

rmse_pop <- calcPredictionAccuracy(prediction_pop, getData(eval, "unknown"))[1]
rmse_pop
```

```
## RMSE
## 0.8482917
```

User based cosine factorization recommender engine

```
#Estimating rmse for UBCF using Cosine similarity and selected n as 50 based on cross-validation
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
```

used

```
model <- Recommender(getData(eval, "train"), method = "UBCF",
param=list(normalize = "center", method="Cosine", nn=50))

prediction <- predict(model, getData(eval, "known"), type="ratings")

rmse_ubcf <- calcPredictionAccuracy(prediction, getData(eval, "unknown"))[1]
rmse_ubcf
```

RMSE
0.8589153

Item based Cosine factorization recommender engine

```
#Estimating rmse for IBCF using Cosine similarity and selected n as 350 based on cross-validation
set.seed(1, sample.kind = "Rounding")
```

Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
used

```
model <- Recommender(getData(eval, "train"), method = "IBCF",
param=list(normalize = "center", method="Cosine", k=350))
prediction <- predict(model, getData(eval, "known"), type="ratings")
rmse_ibcf <- calcPredictionAccuracy(prediction, getData(eval, "unknown"))[1]
rmse_ibcf
```

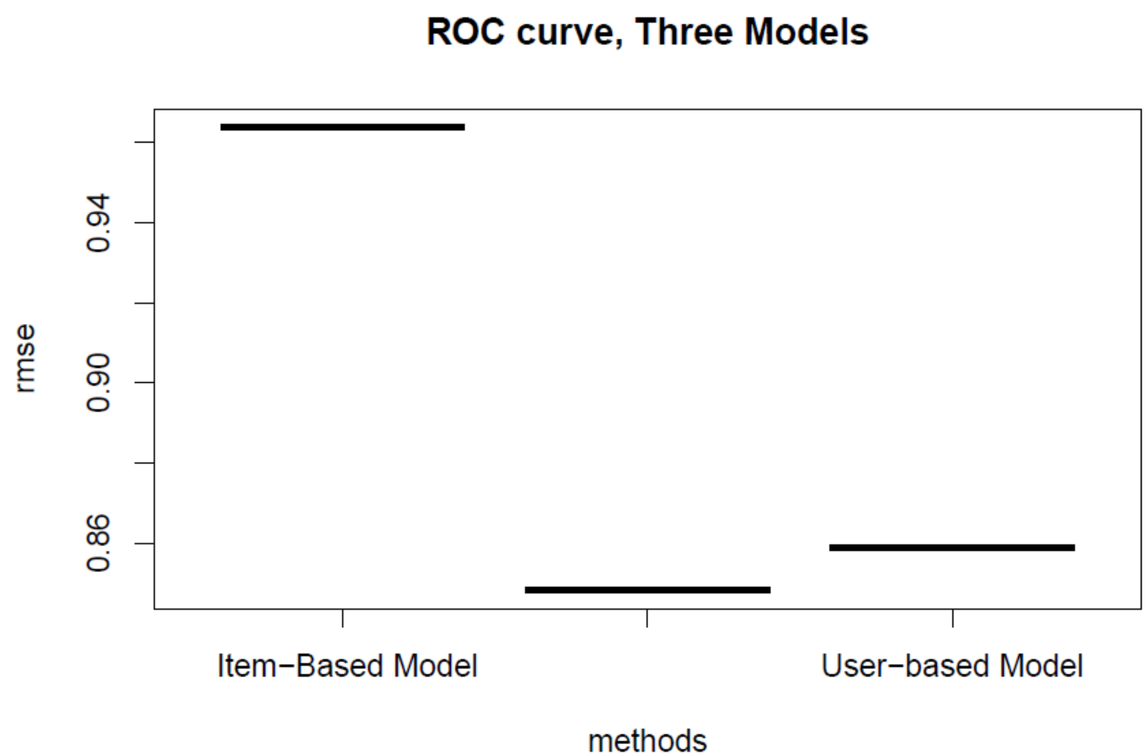
RMSE
0.963769

Rmse from popularity, user and item based recommender engine

```
rmse_crossvalidation <- data.frame(methods=c("Popularity-Based model","User-based Model"," Item-
Based Model"),kable(rmse_crossvalidation) %>%
kable_styling(bootstrap_options = "striped" , full_width = F , position = "center") %>%
kable_styling(bootstrap_options = "bordered", full_width = F , position ="center") %>%
column_spec(1,bold = T ) %>%
column_spec(2,bold =T ,color = "white" , background ="#D7261E")
```

methods	rmse
Popularity-Based model	0.8482917
User-based Model	0.8589153
Item-Based Model	0.9637690

```r
plot(rmse_crossvalidation, annotate = 1, legend = "topleft")
title("ROC curve, Three Models")
```



H. Regularization

The best model recommended from this research from the validation set is the regularized Movie + User effect Model with the least root mean square of 0.8648170.