am5696_Assignment_3

Aryan Mishra

3/2/2022

Importing Relevant Libraries

```
set.seed(1)

library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(reshape2)

setwd('/Users/aryan/Desktop/MSBA_Spring_22/MRKT_9653_ML/Homework_3')
```

Question 1

```
data <- read.delim("/Users/aryan/Desktop/MSBA_Spring_22/MRKT_9653_ML/Homework
_3/BX-Book-Ratings.csv", sep=";")

user_id <- data$User.ID
table_user_ids <- table(user_id) #Frequency of each user ID
table_user_ids_sorted <- table_user_ids[order(table_user_ids,decreasing = TRU
E)] #Frequency table of the most active users
top_100_users <- head(table_user_ids_sorted,100) #100 most active users (with
the most ratings)
df_frequency_table <- as.data.frame(top_100_users) #Display in DF format
head(df_frequency_table)

##   user_id Freq
## 1  198711 7550
## 2  212898 4785
## 3  278418 4533
## 4  235105 3067
## 5  230522 2991
## 6  234623 2674
```

```
data_reduced <- data[which(data$User.ID %in% df_frequency_table$user_id),] #S
ub-setting original dataframe with the 100 most active users.
head(data_reduced)
```

```
##      User.ID       ISBN Book.Rating
## 4331  278418 0006128831           0
## 4332  278418 0006542808           5
## 4333  278418 0020209606           0
## 4334  278418 0020418809           0
## 4335  278418 0020420900           0
## 4336  278418 002043300X           0
```

Our now modified data frame has 134,906 observations. We can verify if this is the correct number of observations by summing the frequencies of our 100 most active users.

```
sum(df_frequency_table$Freq)
```

```
## [1] 134906
```

Indeed, we are on the right path. However, to find the number of unique items/book ratings, we have to find the number of unique ISBN's in our now modified dataframe. This is done using the following code below.

```
print(length(unique(data_reduced$ISBN)))
```

```
## [1] 85118
```

As seen above, we have 85118 unique books that were reviewed by our top 100 users.

Question 2

```
#Train-Test Split
set.seed(1)
train <- sample(1: nrow(data_reduced), size=100000, replace = FALSE)
train_set <- data_reduced[train,] #Training Set in DF format
test_set <- data_reduced[-train, ] #Test set in DF format
test_set_copy <- test_set
```

Question 3: Creating a Ratings Matrix

```
#First we will convert our test values to NA (ratings to be predicted)
test_set$Book.Rating <- NA

# We then need to union the training and test set so we can create our rating
```

```
s matrix
pre_matrix_train_data <- union(test_set, train_set)
ratings_matrix_train <- dcast(pre_matrix_train_data, pre_matrix_train_data$Us
er.ID~ISBN,  value.var=c("Book.Rating"),fill = NA) #Converting dataframe to w
ide form

colnames(ratings_matrix_train)[1] <- "User" #Setting Column name
rownames(ratings_matrix_train) <- ratings_matrix_train$User #Setting Row Name

#Removing first column while creating the matrix
ratings_matrix_train <- ratings_matrix_train[, -1]
ratings_matrix <- data.matrix(ratings_matrix_train)
```

Question 4: SVD

```
row_means <- rowMeans(ratings_matrix, na.rm=TRUE) #Calculating row means to r
eplace the NA values
na_indices <- which(is.na(ratings_matrix), arr.ind=TRUE) #Finding indices of
NA values
ratings_matrix[na_indices] <- row_means[na_indices[,1]] #Replacing NA values
with row means

#Using base function SVD on ratings_matrix
SVD <- svd(ratings_matrix)

#Performing modifications and creating predictions matrix and dataframe
is_this_R1 <- SVD$u %*% diag(SVD$d) %*% t(SVD$v)
predictions_matrix <- round( SVD$u[, 1:2] %*% diag(SVD$d)[1:2, 1:2] %*% t(SVD
$v [, 1:2 ]) ,0) #Predicted ratings (rounded)
predictions_matrix_unrounded <- round( SVD$u[, 1:2] %*% diag(SVD$d)[1:2, 1:2]
%*% t(SVD$v [, 1:2 ]) , 4) #unrounded (rounded to 4 decimal places)
df_matrix <- as.data.frame(predictions_matrix) #Converting predictions matrix
to a data frame for easier manipulation
df_matrix_unrounded <- as.data.frame(predictions_matrix_unrounded)
row.names(df_matrix) <- row.names(ratings_matrix_train) #Adding row name to d
ataframe
colnames(df_matrix) <- colnames(ratings_matrix_train) #Adding column name to
dataframe
row.names(df_matrix_unrounded) <- row.names(ratings_matrix_train)
colnames(df_matrix_unrounded) <- colnames(ratings_matrix_train)
df_matrix$User.ID <- rownames(df_matrix) #Adding user ID to be able to merge
with test set
df_matrix_unrounded$User.ID <- rownames(df_matrix_unrounded)

molten_df_matrix <- melt(df_matrix, id = "User.ID") #Converting dataframe obj
ect to molten data frame
molten_df_matrix_unrounded <- melt(df_matrix_unrounded, id = "User.ID")
colnames(molten_df_matrix)[2] <- "ISBN"
colnames(molten_df_matrix_unrounded)[2] <- "ISBN"
```

```
#Merging with original test set so we can compare our results
final_results<- merge(x = test_set_copy, y = molten_df_matrix, by.x = c("User
.ID", "ISBN"), by.y = c("User.ID", "ISBN"), all.x = TRUE)
final_results_unrounded <- merge(x = test_set_copy, y = molten_df_matrix_unro
unded, by.x = c("User.ID", "ISBN"), by.y = c("User.ID", "ISBN"), all.x = TRUE
)

#Calculating RMSE
RMSE <- sqrt(mean((final_results$value - final_results$Book.Rating)**2))
RMSE

## [1] 2.568285
```

As seen above, our recommendation system has a RSME of 2.568285. RMSE can be interpreted as the standard deviation of the standard deviation of the unexplained variance. The lower the value, the better the fit. While it is hard to examine the model's performance without comparing it with another model, based on a rule of thumb, RMSE values between 0.2 and 0.5 show that the model can relatively predict the data accurately. However, our RMSE is quite high at 2.56825, indicating that our model is not performing very well and there are plenty of ways we can improve it.

Question 5: Iterative SVD

```
# Recreating Ratings Matrix using Same Steps as Before
Round_2 <- SVD$u[, 1:2] %*% diag(SVD$d)[1:2, 1:2] %*% t(SVD$v [, 1:2 ])
dataset_iteration_2<- data.frame(final_results_unrounded$User.ID, final_resul
ts_unrounded$ISBN, final_results_unrounded$value)
colnames(dataset_iteration_2) <- c("User.ID", "ISBN", "Book.Rating")
training_data_unrounded <- union(dataset_iteration_2, train_set)
ratings_matrix_2_train <- dcast(training_data_unrounded, training_data_unroun
ded$User.ID~ISBN, value.var=c("Book.Rating"), fill = NA)
colnames(ratings_matrix_2_train)[1] <- "User"
rownames(ratings_matrix_2_train) <- ratings_matrix_2_train$User
ratings_matrix_2_train <- ratings_matrix_2_train[, -1]
ratings_matrix_2_unrounded <- data.matrix(ratings_matrix_2_train)
na_indices_2 <- which(is.na(ratings_matrix_2_unrounded), arr.ind=TRUE)
ratings_matrix_2_unrounded[na_indices_2] <- rowMeans(ratings_matrix_2_unround
ed, na.rm=TRUE)[na_indices_2[,1]] #Replacing NAs with row means

# Running SVD again & Evaluating Test Set Performance

SVD <- svd(ratings_matrix_2_unrounded)
R3 <- round( SVD$u[, 1:2] %*% diag(SVD$d)[1:2, 1:2] %*% t(SVD$v [, 1:2 ]), 0)
df_matrix_2 <- as.data.frame(R3)
row.names(df_matrix_2) <- row.names(ratings_matrix_2_train)
colnames(df_matrix_2) <- colnames(ratings_matrix_2_train)
```

```
df_matrix_2$User.ID <- rownames(df_matrix_2)
molten.df_matrix_2 <- melt(df_matrix_2, id = "User.ID")
colnames(molten.df_matrix_2)[2] <- "ISBN"
#Merging with original test set so we can compare our results
final_results_2 <- merge(x = test_set_copy, y = molten.df_matrix_2, by.x = c(
"User.ID", "ISBN"), by.y = c("User.ID", "ISBN"), all.x = TRUE)
#Calculating RMSE
RMSE <- sqrt(mean((final_results_2$value - final_results_2$Book.Rating)**2))
RMSE

## [1] 2.568006
```

Using the iterative SVD, we get a slightly better value at 2.568006. Compared to our original model, the Iterative SVD model is better since its RMSE is slightly lower. The reason for this could be because performing SVD when the rankings matrix is sparse (like in our case) can be a challenge, so using an iterative procedure where we perform rank-2 SVD is the reason why we end up yielding better results. However, there is still a lot more room for improvement.

Question 6: Improvements

As with many machine learning algorithms, SVD has a lot of moving parts. One way we can improve model performance is to tune some of the hyper parameters such as the number of factors. There are several other solutions to improving the performance of our model. Getting more ratings is always useful as this means more data for our model to train on. Furthermore, it is important to note that our data set (like most of the other recommendation data sets) has very few books that contain the majority of the ratings. Most of the books have very few ratings which affects the accuracies of the predictions and will generally be different on the popular books than the sparsely rated books. A solution to this problem is to give book specific weights toward each of the books when computing RSME. These could allow for a more accurate representation of how the recommender system is evaluating all of the items.

References

https://www.geosci-model-dev.net/7/1247/2014/gmd-7-1247-2014.pdf Chapter 7 of Recommender Systems: The Textbook by Charu Aggarwal.