Recommender Systems

6/24/2017

DATA643 Project 3

by Tulasi Ramarao

Description

In this project, a data analysis tool called Singular Value Decomposition(SVD) will be used to create the recommender. SVD was used by BellKor team in their recommender project that won the Netflix Prize in 2009. SVD is a matrix factorization technique used to reduce the number of features of a given dataset by reducing dimensions and the matrix factorization here is performed on a user-item ratings matrix.

Dataset

The dataset from https://grouplens.org/datasets/movielens/ listed under Recommended for Education and Research is downloaded. This dataset contains 100,000 ratings for 9,000 movies by 700 users. The ratings are on the scale of 0-5.

Note

This project is coded to handle any kind of user-item ratings data. For example- movies, beer, books, fashion clothes, etc. This project uses a custom function written to get ratings using irlba and the full svd methods. This function also handles errors. The other functionality is that this getrating function uses sqldf to query against the movies file to get the name of the movies. This will help in displaying a meaningful message.

Data preparation:

The data is loaded from a csv file and the first columns are chosen.

```r
set.seed(3445) # to keep #s from the results the same

# Set the working directory
setwd("/Users/tulasiramarao/Documents/Tulasi/CUNYProjects/DATA643/RPrograms")

# load data from local drive
dfratings <- read.csv("MovieRatingsData/ml-latest-small/ratings.csv", header = TRUE, sep =",",
                     stringsAsFactors = FALSE)

colnames(dfratings)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp"
```

```r
#Select the relevant columns by using numbers to generalize the subset method for other datasets.
myratings <- subset(dfratings,select = c(1,2,3))
```

The function dplyr is used to reformat data, making rating to be the values in the matrix; user as rows, movies as columns. Also the dataframe is converted to a matrix format to feed to the recommenderlab library.

```r
reformatted.df <- myratings %>%
  spread(key = movieId, value = rating) %>%
  as.matrix()

# Remove first column - userid ( rows are users so no need for userid)
mydf = reformatted.df[,-1]  # Note: Not using subset to make this a generalized code
```

Next, the matrix size is reduced by converting the matrix to a readRating matrix (from the recommenderlab) and coerced into a realRatingMAtrix. This will greatly reduce the file size. Data is prepared based on instructions under Chapter 3 of "Building a recommendation system with R" - from Suresh Gorakala)

The movies dataset contains movies that have been viewed only a few times with ratings biased because of the lack of datapoints. Also the users rated a few movies making the ratings biased also. So its necessary to have a required minimum of users and movies. So we will define ratings_movies containing the matrix - with users that have rated at least 10 movies

```r
# also using realRatingMatrix from recommenderlab to reduce dimensions
mydf <- as(mydf, "realRatingMatrix")
dim(reformatted.df)
```

```
## [1]  671 9067
```

The following plot displays the histogram of the ratings on the scale of 0 to 5 (5 being the best).

```r
vector_ratings <- as.vector(dfratings$rating)
unique(vector_ratings)
```

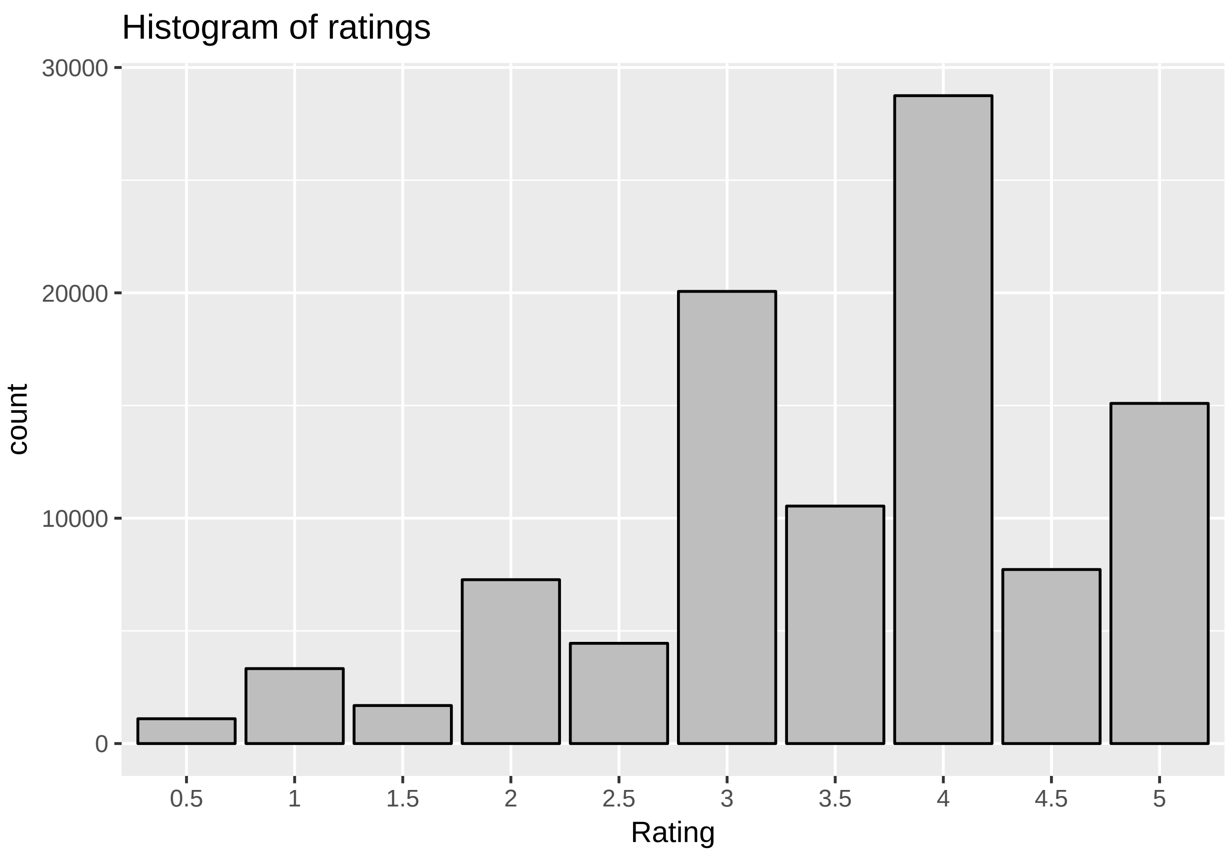
```
##  [1] 2.5 3.0 2.0 4.0 3.5 1.0 5.0 4.5 1.5 0.5
```

```r
# The ratings are in the range 0-5. Count the occurences of each of them.
table_ratings <- table(vector_ratings)
```

According to the documentation, a zero rating represents a missing value, so it was removed from vector_ratings.

```r
vector_ratings <- factor(vector_ratings)

qplot(vector_ratings, main ="Histogram of ratings", xlab = "Rating",fill=I("grey"),col=I("black"))
```



```
summary(vector_ratings)
```

```
##   0.5     1   1.5     2   2.5     3   3.5     4   4.5     5
##  1101  3326  1687  7271  4449 20064 10538 28750  7723 15095
```

```
# Movies rated > 10 times
ratings_movies <- mydf[,colCounts(mydf) > 10]

# Creates a 671 x 2083 rating matrix of class 'realRatingMatrix' with 80295 ratings.
#head(ratings_movies,3)
#head(mydf)

# group data by the item and then the user
itemStats <- myratings %>% group_by(userId) %>% group_by((movieId)) %>%
              summarise(
                review_avg = mean(rating)
              )
```

Reshape the dataframe so that the users are in the rows and the item names are in the column.

```
# Generating the user-item matrix
user_item <- dcast(myratings, userId~movieId,
                   value.var = "rating", fill=0, fun.aggregate = mean)

# Filling in rownames
rownames(user_item) = user_item$userId

# Removing the first column
user_item <- user_item[,-1]
```

```
# Converting to a matrix
user_item <- as.matrix(user_item)
```

Singular Value Decomposition (SVD)

The matrix factorization technique called Singular value decomposition (SVD) is used to reduce dimensionality, so neighborhood formation happens in a reduced user space. SVD helps the model to find the low rank dataset matrix. However, SVD is computationally expensive. So the irlba, a package from R will be used to perform the *singular value decomposition*. The irlba function computes only the number of singular values corresponding to the maximum of the desired singular vectors, max(nu, nv). For example, if 5 singular vectors are desired (nu=nv=5), then only the five corresponding singular values are computed. In contrast, SVD without irlba function always returns the total set of singular values for the matrix, regardless of how many singular vectors are specified.

The *irlba* package provides a fast way to compute partial singular value decompositions (SVD) of large sparse or dense matrices. The paramters to irlba are: nv = number of right singular vectors to estimate nu = number of left singular vectors to estimate

Performance testing:

First perform irlba and calculate the processing time.

```
# Use tic toc to measure time
tic()
# Feed the matrix form of data
irlba.form = irlba(user_item, nu = 3, nv = 3)
exectime <- toc()
```

```
## 0.202 sec elapsed
```

```
exectimeIrlba <- exectime$toc - exectime$tic
```

Now the full SVD is used to calculate the processing time.

```
tic()
#Feed the matrix form of data
svd.form <- svd(user_item,nu=3,nv=3)
exectime <- toc()
```

```
## 11.076 sec elapsed
```

```
exectimeSVD <- exectime$toc - exectime$tic
```

When these processing times are compared, irlba performanced 58 times better than svd.

Prediction

Now predicting the ratings using the irlba prediction.
For this, results from the irlba call are used, namely, u: nu approximate left singular vectors v: nv approximate right singular vectors d: max(nu,nv) approximate singular values

First the average of user reviews are calculated and added to the rest of the values in the irlba calculation.

```
irlba.value <- (irlba.form$u * sqrt(irlba.form$d)) %*% (sqrt(irlba.form$d) * t(irlba.form$v))

# and then calculate the prediction for irlba
```

```
predict.irlba <- itemStats$review_avg + irlba.value
# Need column/row names for predicting user and item later.
colnames(predict.irlba) <- colnames(user_item)
rownames(predict.irlba) <- rownames(user_item)
```

Now, predicting the ratings using the regular SVD prediction:
Its parameters are:
u: a matrix whose columns contain the left singular vectors of X
v: a matrix whose columns contain the right singular vector of X
d: a vector containing the singular values of x

```
# Error: longer object length is not a multiple of shorter object length
#svd.value <- (svd.form$u * svd.form$d) %*% (svd.form$d * t(svd.form$v))

val1 <- sqrt(diag(svd.form$d[1:3]))
svd.value <- (svd.form$u %*% val1) %*% ( val1 %*% t(svd.form$v))
# and then calculate the prediction for irlba
predict.svd <- itemStats$review_avg + svd.value

# Need column/row names for predicting user and item later.
colnames(predict.svd) <- colnames(user_item)
rownames(predict.svd) <- rownames(user_item)
```

RMSE

Most popular measures of accuracy are Precision/Recall and RMSE. Here, the RMSEs of SVD and irlba will be compared to decide which is a good evaluator of a recommender system. RMSEs are the differences between the actual and the predicted values and can either be positive or negative in value as the predicted value under or over estimates the actual value.

First, the RMSE is calculated for irlba -

```
# exclude missing values NA from analysis with na.rm = True
(RMSE.irlba <-  sqrt(mean((predict.irlba - user_item)^2, na.rm=T)))
```

```
## [1] 3.421318
```

Then the RMSE value for SVD is calculated -

```
# exclude missing values NA from analysis with na.rm = True
(RMSE.svd <-  sqrt(mean((predict.svd - user_item)^2, na.rm=T)))
```

```
## [1] 3.43567
```

There is only a slight improvement in RMSE, when the irlba package is used.

Prediction:

Now a predicted rating is calculated from a custom function created for this purpose. This function takes in three parameters - the user-name, the item-name and the methodology(irlba or SVD). It also checks for any exceptions(error checking). If there is an error in finding the user or the item in the matrix, informs the users that they need to enter valid values again.

```
dfmovies <- read.csv("MovieRatingsData/ml-latest-small/movies.csv", header = TRUE, sep = ",",
                     stringsAsFactors = FALSE, encoding = "UTF-8")
names(dfmovies)
```

```
## [1] "movieId" "title"   "genres"
```

```
data1 <- sqldf("select title from dfmovies where movieId = 5816")
```

```
## Loading required package: tcltk
```

```
movieID1 = 5816
sql <- sprintf("select title from dfmovies where movieId = %s ", movieID1)

querya <- sqldf(sql)
```

A custom function is created to predict rating of a given user. The method is enclosed in a tryCatch clause to catch exceptions. If the rating exists, then that rating value is returned from the fucntion. Then the logic splits for irlba and svd methods. If an irlba method is requested, the irlba package is used and if svd method is requested, the svd package is used.

The sqldf library is used to query data and to join tables.

```
getRating <- function(user,item,method){
  # Handle errors like this -
  # Error in user_item[user, item] : subscript out of bounds
  tryCatch(
     # If rating exists already
    if(user_item[user,item] !=0){
      paste(method, " Existing rating: ", round(user_item[user,item], 1))
    }
    else{ # if irlba or svd
     if(method == "irlba"){
        sql <- sprintf("select title from dfmovies where movieId = %s ", item)
        querya <- sqldf(sql)
        paste(method, " method: Predicted rating from user#:", user, " for ",
              querya, " is ",   round(predict.irlba[user,item],1))
      }else if(method == "svd"){
        sql <- sprintf("select title from dfmovies where movieId = %s ", item)
        querya <- sqldf(sql)
        paste(method, " method: Predicted rating from user#:", user, " for " ,
              querya, " is ", round(predict.svd[user,item],1))
      }else{
        paste(method, " Please enter user and item numbers ")
      }
  }
  ,error = function(e) print("The requested data does not exist.
                              Please enter valid data for user/item."))
}
```

Testing:

First test this function with false data to test the error handling section of the code. Code is commented out to enable knitting of this document into a .pdf document.

```
#getRating("555", "Hello","irlba")
```

The getRating call for non-existing data returns this error message: "Requested Data does not exist. Please enter a different user-item data".

Now predict a rating for existing data, choosing either irlba or svd function calls.

```
# choosing svd
getRating("2", "5459","svd")
```

```
## [1] "svd  method: Predicted rating from user#: 2  for  Men in Black II (a.k.a. MIIB) (a.k.a. MIB 2)
```

Now call the function

```
# choosing irlba
method = "irlba"
ratingval <- "5"
itemval <- "5816"
userval <- "3"

PredictA <- getRating(userval,itemval,method)   # harry potter
mylist <- strsplit(PredictA,"[:]")
pA <- sapply(mylist,tail,1)

# Check the movie titles of user# 3's highest/5 ratings
sql <- sprintf("select genres from dfratings,dfmovies where dfratings.movieId = dfmovies.movieId and df
queryresults <- sqldf(sql)

resultsA<- as.data.frame(queryresults)
pA <- as.data.frame(PredictA)
kable(pA)
```

PredictA

irlba method: Predicted rating from user#: 3 for Harry Potter and the Chamber of Secrets (2002) is 0.8

```
strA<- paste("User #:", userval," gave 5 ratings for genres shown below")
strA
```

```
## [1] "User #: 3  gave 5 ratings for genres shown below"
```

```
kable(resultsA)
```

genres

Crime|Drama
Comedy|Drama|Romance|War
Action|Adventure|Comedy|Fantasy|Romance Action|Crime|Drama|Thriller
Drama

The resulting rating make sense because the user seems to prefer comedy and romance and Harry potter's movie from Adventure Genres is not likely to get a high rating from this user.

Findings and Recommendations

The concept of SVD(Singular Value Decomposition) was explored in this project. SVD is a form of dimensionalty reduction technique to extract the maximum variability in observations. It's sometimes called unsupervized learning where labeled observations are not required and groups of observations that are similar are found by clustering. Reducing dimensions without losing important information was achieved by using a library called irlba. The performance of irlba package was compared with the svd package and was found that irlba performed much better. The recommender was then created with custom methods to predict a rating of a given user. The results were then analyzed for accuracy.

The drawbacks of using SVD are that the ratings in the observations should not have any missing value. In these situations, scaling the data is almost the best practice (Ref#3). Imputing the values to solve the missing values in a sparse matrix was done in this project by using the irlba package. Eventhough imputing the missing values was computationally expensive, once the initial calculations are done, the other calculations is fast. So, incrementally additional features can be added with some limitations. Adding too many features can decay the performance of the recommender.

Other SVD methods like PCA(Principle Components Analysis) and other packages like xgboost to find the mean centered values for missing ratings were not explored in this project.

References:

Ref#1: https://medium.com/@m__n__malaeb/singular-value-decomposition-svd-in-recommender-systems-for-non-math-statist
Ref#2: https://github.com/dstern04/MSDA-Work/blob/master/661%20-%20Recommender%20Systems/Project.rmd
Ref#3: Andy Catlin's Meetup 3 lecture.
Ref#4: Buiding a recommendation System with R - Suresh Gorakala,Michele Usuelli