title: "MovieLens Project – Capstone Question 1" author: "Saikat Banerjee" date: "7 April 2019" output: pdf_document: default html_document: default

MovieLens Project – Capstone Question 1

Data Science: Capstone edX – HarvardX

Saikat Banerjee

INTRODUCTION

THE DATASET

MovieLens is a project of GroupLens Research. It is a research lab based in the Department of Computer Science and Engineering at the University of Minnesota. It has been present since 1997. The project's main focus was to gather data for research on personalized recommendations. MovieLens data is used to develop the first ever automated recommender system.

The Movielens dataset is used to predict a user's ratings for a particular movie based on the same user's ratings of other movies. Another way to do this is to assume that users who have rated the same movie in a similar way will rate highly corelated movies in a similar fashion as well.

In recommendation system design, these two techniques are known as User Based Collaborative Filtering (UBCF) and Item Based Collaborative Filtering (IBCF).

This dataset is really interesting and the recommendation system design is one of the hottest topics in the current times. Every online platform has some kind of recommendation system implemented in the backend. Starting from Amazon, Netflix even hotel booking sites like Expedia is working on building better recommendation systems so as to provide their customers personalized recommendations to suit their needs.

ANALYSIS

THE EXPLORATORY DATA ANALYSIS

I start by downloading the ratings data from Grouplens.org site.

```
# Downloading the data locally
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

# Reading the temporarily downloaded data into ratings
ratings <- read.table(text = gsub("::", "\t", readLines(unzip(dl, "ml-
10M100K/ratings.dat"))),
                      col.names = c("userId", "movieId", "rating", "timestamp"))
```

Then I do some exploratory data analysis to understand the dataset better.

I use a few of the common functions to do this.
• str
• head
• summary
• table

The str(ratings) tells that the variable Dataset is not a Factor but is a int in the dataset in its current form

```
> str(ratings)

'data.frame':   10000054 obs. of  4 variables:
 $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ movieId  : int  122 185 231 292 316 329 355 356 362 364 ...
 $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
 $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392
838984474 838983653 838984885 838983707 ...
```

I use the head(ratings) to look at a few data points in the dataset

```
> head(ratings)

  userId movieId rating timestamp
1      1     122      5 838985046
2      1     185      5 838983525
3      1     231      5 838983392
4      1     292      5 838983421
5      1     316      5 838983392
6      1     329      5 838983392
```

I try to understand the distribution of variable rating.

```
> table(ratings$rating)

  0.5       1     1.5       2     2.5       3     3.5       4     4.5       5
  94988  384180  118278  790306  370178 2356676  879764 2875850  585022 1544812
```

I try to get some statistics on my dataset using the summary function

```
> summary(ratings)

     userId         movieId          rating        timestamp
 Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
 1st Qu.:18123   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
 Median :35741   Median : 1834   Median :4.000   Median :1.035e+09
 Mean   :35870   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
 3rd Qu.:53608   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
 Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
```

In building my recommendation system, I do not see the need for the column timestamp. So subsequently, I will be building a dataframe without the timestamp column.

I then download the movies data and merge it with my ratings data to get the movielens dataset. This dataset is the basis of all the datasets I will be working on.

```
# Reading the data into movies
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::",
3);

# Defining column names - movieId, title and genres
colnames(movies) <- c("movieId", "title", "genres");

# Populating the data in the respective column names
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))
[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres));

# Doing a left join between ratings and movies on the field "movieId"
movielens <- left_join(ratings, movies, by = "movieId")
```

It is important to note that there are no missing values in both my datasets

```
> sum(is.na(ratings$rating))
[1] 0
>
> sum(is.na(movielens$rating))
[1] 0
```

It is important to note there are no missing data

```
> colSums(sapply(movielens, is.na))
  userId   movieId    rating timestamp     title    genres
       0         0         0         0         0         0
```

CREATION OF edx AND validation DATASETS

```
# Validation set will be 10% of MovieLens data

set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.10, list
= FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Making sure userId and movieId in validation set are also in edx set

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

I will be using recosystem library. This library builds recommendation system using the Matrix Factorization technique. This library has the dependency on the library Rcpp library. So it is recommended that while installing the library recosystem, please set the option Dependency as True. For more details, you can read the following link.

https://cran.r-project.org/web/packages/recosystem/recosystem.pdf

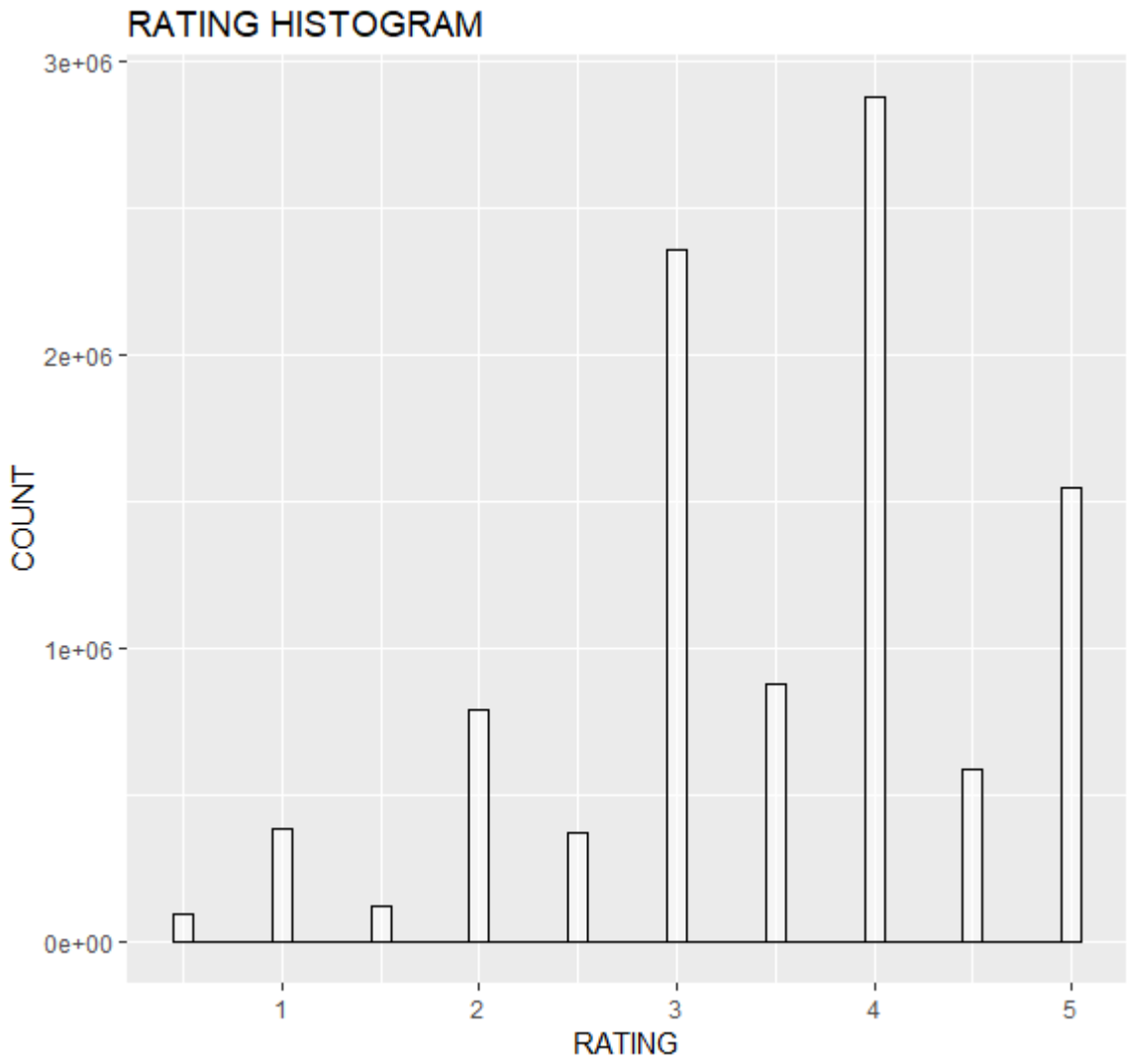
DATA VISUALIZATION

I try to visualize the spread of total numbers of ratings across each rating category. I run the following code on my ratings dataset.

```
# Trying to visualize the ratings data

ggplot(ratings, aes(x=rating)) +
  geom_histogram(binwidth=0.1,color="black", fill="white",alpha=0.5)+
  labs(title="RATING HISTOGRAM",x="RATING", y = "COUNT")
```

And the following is observed:



MODELING APPROACH

First, I break down the dataset into train and test datasets. The createDataPartition function from the caret package is used to do this.

```
# Validation set will be 10% of MovieLens data

set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.10, list
= FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Making sure userId and movieId in validation set are also in edx set

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

So, I use 90% of my data as the trainset while the rest 10% I use as my test set.

Now to use the recommender built by the Reco() function in the recosystem library, I need to build my train_set and test_set in the proper format.

```
#Creating the datasets in proper format for running the recommendation using
recosystem library

train_data <- data_memory(user_index = edx$userId, item_index = edx$movieId,
                          rating = edx$rating, index1 = T);

test_data <- data_memory(user_index = validation$userId, item_index =
validation$movieId, rating = validation$rating, index1 = T);
```

Please note here, that train_data is created from edx set and test_data is created from validation set. Also note that in the data userId and movieId both start from 1. So, I have set index1 = T here.

These two datasets train_data and test_data are in the format which can be used with the function from recosystem library.

My model looks like the following:

```
#Creating the recommender using recosystem library
recommender <- Reco()

#Training the recommender on train_data which is created from edx dataset
# opts argument is a list to supply the number of parameters
# dim is an integer which represents the number of latent factors
# costp_l2 is a numeric L2 regularization parameter for user factors
# costq_l2 is a numeric L2 regularization parameter for item factors
# lrate is numeric learning rate
# niter is an interger representing number of iterations
# nthread is an integer representing number of threads for parallel computing
# nbin is an integer representing number of bins, must be greater than nthread -
default is used in this case
# verbose is a logical parameter to show detailed information
# For more info on these tuning parameters try: ?recosystem::train

recommender$train(train_data, opts = c(dim = 30, costp_l2 = 0.1, costq_l2 = 0.1,
                                       lrate = 0.1, niter = 100, nthread = 6,
verbose = F))

#Predicting the ratings on test_data which is created from validation dataset
prediction <- recommender$predict(test_data, out_memory())
```

The recommender is built calling the Reco() function from the recosystem library. After that, I use the $train attribute of the recommender to train my model on my train data. There are options to provide several tuning parameters. I use the parameter values as shown above. If you want to find out more about the parameters that we can use, you may try the following command in R: ?recosystem::train.

Also note, it is imperative to use the parameter out_memory() with the recommender$predict function.

RESULTS

```
# Creating the dataframe for predicted ratings
pred <- data.frame("predicted ratings" = prediction)

# Creating the dataframe with actual ratings from the validation set
test_data_df <- data.frame("actual ratings"=validation[,c(3)])

# Calculating the RMSE
RMSE <- RMSE(pred,test_data_df)
```

```
# Outputting the RMSE
# Please note the header is basically the name of the column of the pred dataset

> print(RMSE)
predicted.ratings
        0.8115481
```

```
# Creating the dataframe which I will use to generate my submission file
pred_df <- cbind(validation,pred)

# Writing the submission.csv file
# The file contains the userId,movieId and corresponding predicted ratings
write.csv(pred_df %>% select(userId, movieId, "predicted.ratings"),
"submission.csv", na = "", row.names=FALSE)
```

Once we get the prediction, we need to put it into dataframes that we can work with to calculate the RMSE.

RMSE is calculated using the RMSE function of the SimDesign library. The first parameter of the function is the estimate and second parameter is the original dataset.

The RMSE for my model is **0.811476**

The results with UserId, MovieId and PredictedRatings in the .csv format can be found here:

https://docs.google.com/spreadsheets/d/1sJo4yskT_Y0qmORIOywlTHjVItV1E2fwmPvVovoqAq8/edit?usp=sharing

CONCLUSION

My model is built using the Matrix Factorization approach implemented in the Recosystem package of R. Recosystem is a R wrapper of the libmf library for recommender system using matrix factorization. It is available as a CRAN Package. It can be used for High Performance, in a distributed multi-core parallel programming environment.

My model has a RMSE of **0.811**.

This RMSE can further improved my optimizing the tuning parameters while training the recommender model. Another approach, could be to combine the recommender model with neighborhood optimization approaches.