MOVIE RECOMMENDATION SYSTEM PROJECT

Reza Ameri, Ph.D.

Contents

1 Abstract

Recommender systems have become prevalent in recent years as they tackle the problem of information overload by suggesting the most relevant products to end users. In fact, recommender systems are information filtering tools that aspire to predict the rating for users and items, predominantly from big data to recommend their likes. Specifically, movie recommendation systems provide a mechanism to assist users in classifying users with similar interests. In this project, exploratory data analysis is used in order to develop various machine learning algorithms that predict movie ratings with reasonable accuracy. The project is part of the capstone for the professional certificate in data science program at Harvard University. The MovieLens 10M Dataset that includes 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users is used for creating a movie recommendation system.

2 Introduction

Fast development of technology and the internet has resulted in a rapid growth of available information over the past few decades. Recommendation systems, as one of the most successful information filtering applications, have become an efficient way to solve the information overload problem [1]. The main goal of a recommendation system is to automatically generate suggested items for end users based on their historical preferences.

Recommendation systems have become an indispensable component in various e-commerce applications [2]. Recommender systems collect information about the user's preferences of different items (e.g. movies, shopping, tourism, TV, taxi) by two ways, either implicitly or explicitly [3-7]. An implicit acquisition of user information typically involves observing the user's behavior such as watched movies, purchased products, downloaded applications. On the other hand, a direct procurement of information typically involves collecting the user's previous ratings or history [2].

In this project, a movie recommendation system is developed using the MovieLens 10M Dataset [8] which includes 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users, collected by the GroupLens Research Lab [9]. Considering five different models, the Mean Square Error (RMSE) is used to evaluate how close the predictions are to the true values in the final hold-out test set. The model with the lowest RMSE is reported as the best model. It should be noted that, in the machine learning models, the final hold-out test set (i.e. the validation data) has not been used for training, developing, or selecting the algorithm, and has been only used for evaluating the RMSE of the final algorithm.

3 MovieLens Dataset

The MovieLens dataset can be downloaded from ref. [8]. Then, the following code can be run in order to generate the datasets:

```
# Create train and validation sets

# 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(recosystem)) install.packages("recosystem", 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")
```

```
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)
```

As previously mentioned, the algorithms will be developed using the train (i.e. edx) set. Movie ratings will be then predicted in the validation set, for a final test of the final algorithm.

4 Data Processing

In order to get familiar with the dataset, the structure of the data as well as its summary are evaluated as shown below:

```
# Structure of the dataset
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ad
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
# Headers of the dataset
head(edx) %>%
  print.data.frame()
```

```
##   userId movieId rating timestamp                       title
## 1      1     122      5 838985046             Boomerang (1992)
## 2      1     185      5 838983525              Net, The (1995)
```

```
## 3      1     292      5 838983421                Outbreak (1995)
## 4      1     316      5 838983392                Stargate (1994)
## 5      1     329      5 838983392 Star Trek: Generations (1994)
## 6      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 3  Action|Drama|Sci-Fi|Thriller
## 4       Action|Adventure|Sci-Fi
## 5 Action|Adventure|Drama|Sci-Fi
## 6       Children|Comedy|Fantasy
```

```r
# Summary of the dataset
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

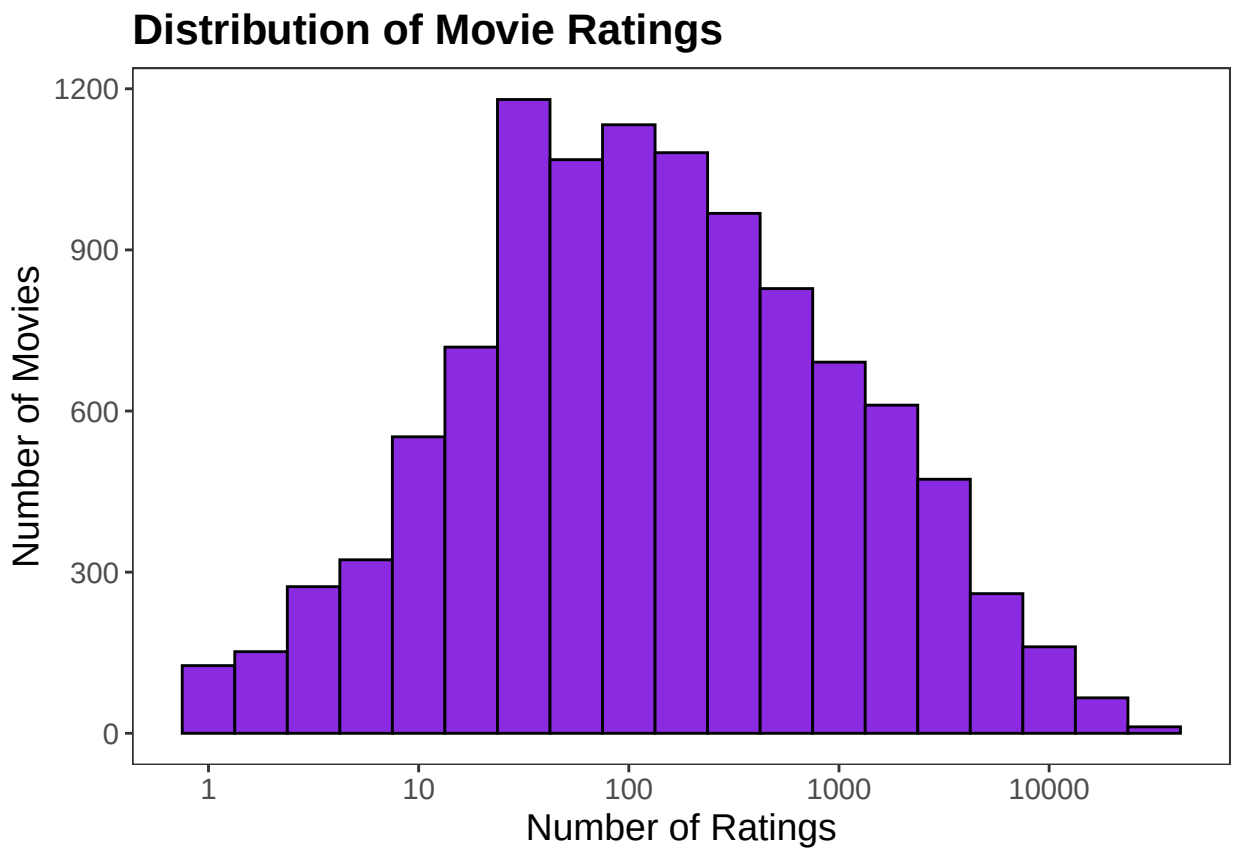
As can be seen, the dataset contains six variables, including “userId”, “movieId”, “rating”, “timestamp”, “title”, and “genres”. The dataset is in the tidy format and ready for processing.

We do know, from the contents of the course, that some movies are generally rated higher than others. We also know that there is substantial variability across users (e.g. some users love every movie and others are very critical). There are some other important sources of variation related to the fact that groups of movies have similar rating patterns and groups of users have similar rating paterns. In order to have a better understanding of the variability in data, several insightful plots can be made as presented below.

First, we can plot the number of ratings per movie using the following code:

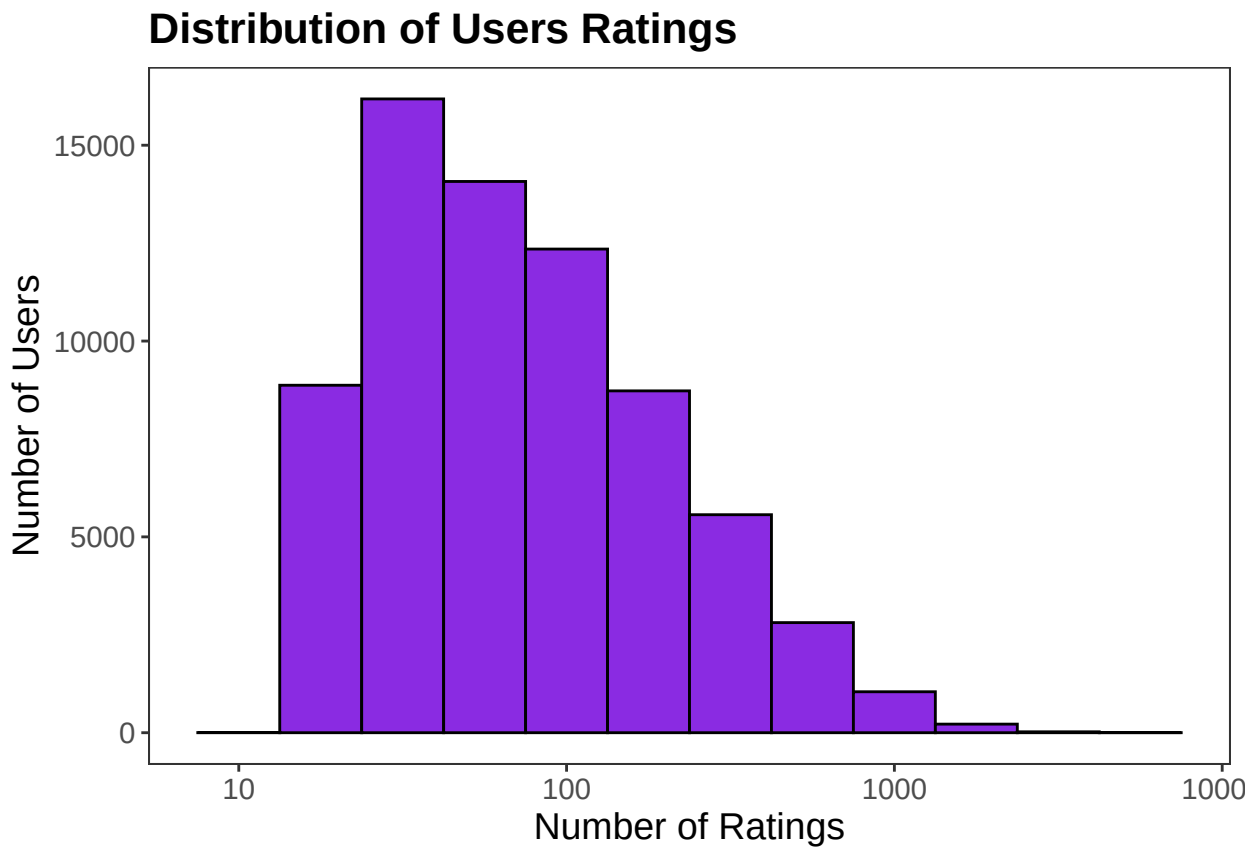
```r
# Ratings per movie distribution
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(binwidth = 0.25, color="black", fill="blueviolet") +
  scale_x_log10() +
  xlab("Number of Ratings") +
  ylab("Number of Movies") +
  ggtitle("Distribution of Movie Ratings") + theme_bw() +
  theme(plot.title = element_text(size = 16, face = "bold"),
        axis.text.y   = element_text(size=11),
        axis.text.x   = element_text(size=11),
        axis.title.y  = element_text(size=14),
```

```
        axis.title.x  = element_text(size=14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background=element_rect(fill = "white"))
```



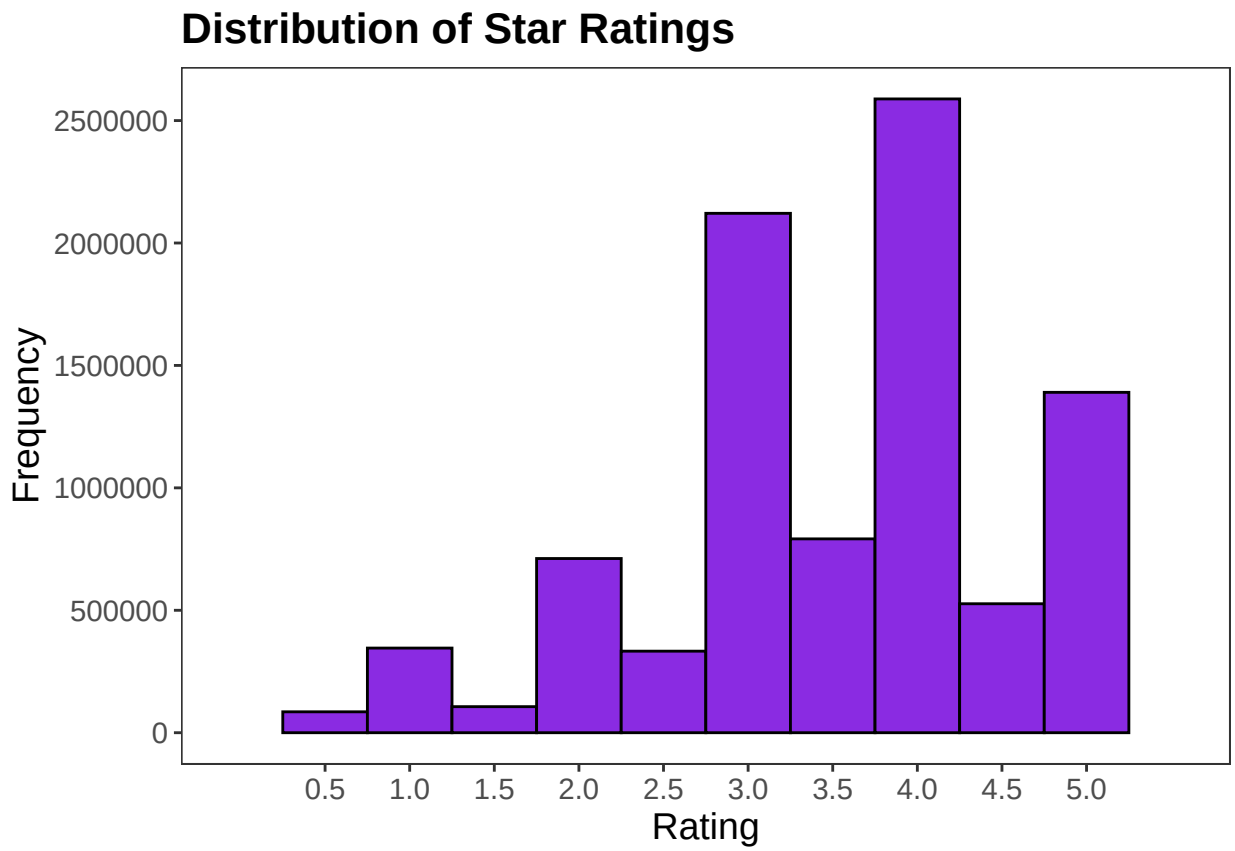
It can be seen that majority of movies have been rated approximately between 50 to 150 times. The number of users' ratings frequency can also be plotted using the following code:

```
# User ratings distribution
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(binwidth = 0.25, color="black", fill="blueviolet") +
  scale_x_log10() +
  xlab("Number of Ratings") +
  ylab("Number of Users") +
  ggtitle("Distribution of Users Ratings") + theme_bw() +
  theme(plot.title = element_text(size = 16, face = "bold"),
        axis.text.y   = element_text(size=11),
        axis.text.x   = element_text(size=11),
        axis.title.y  = element_text(size=14),
        axis.title.x  = element_text(size=14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background=element_rect(fill = "white"))
```



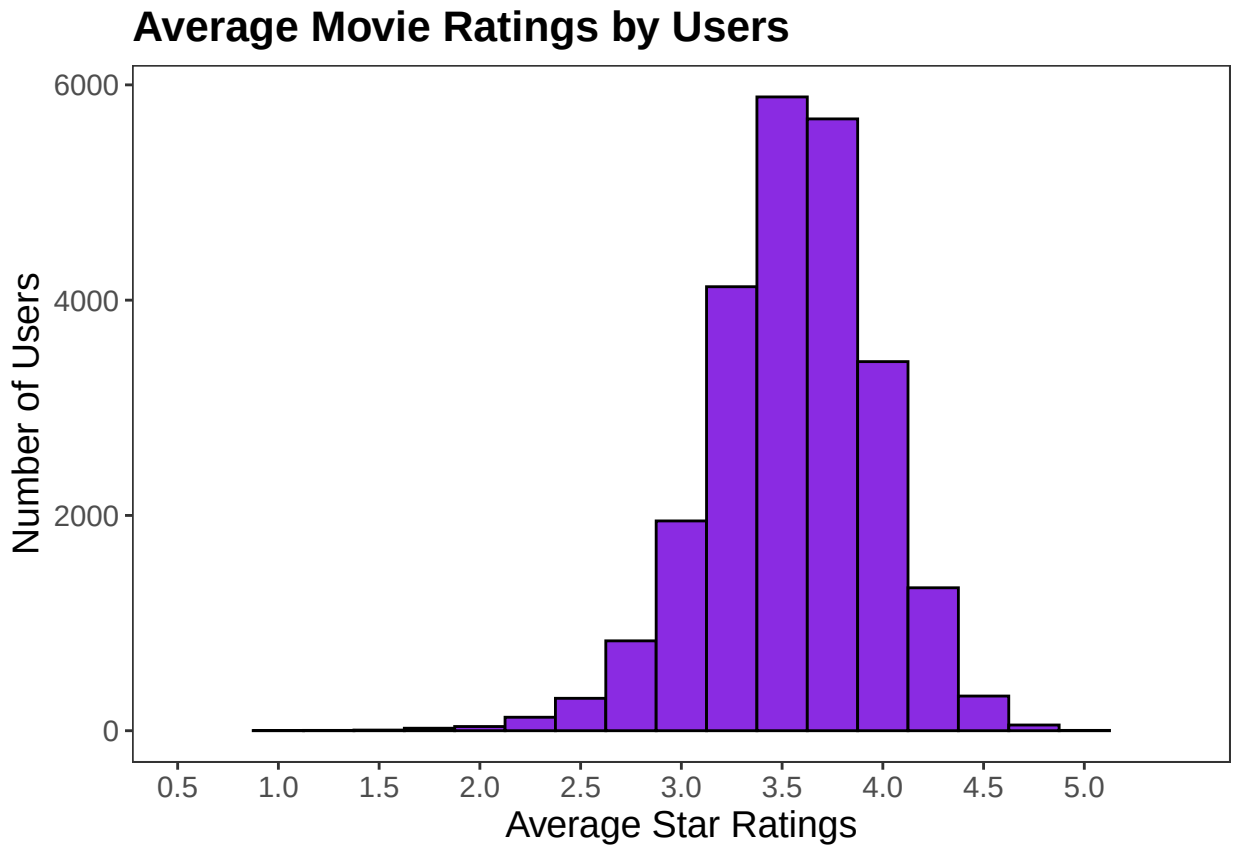
The plot represents that majority of users have rated approximately between 30 to 120 movies. We can use the following code to plot the distribution of star ratings given to movies by users:

```
# Star ratings distribution
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.5, color="black", fill="blueviolet") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +
  xlab("Rating") +
  ylab("Frequency") +
  ggtitle("Distribution of Star Ratings") + theme_bw() +
  theme(plot.title = element_text(size = 16, face = "bold"),
        axis.text.y   = element_text(size=11),
        axis.text.x   = element_text(size=11),
        axis.title.y  = element_text(size=14),
        axis.title.x  = element_text(size=14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background=element_rect(fill = "white"))
```



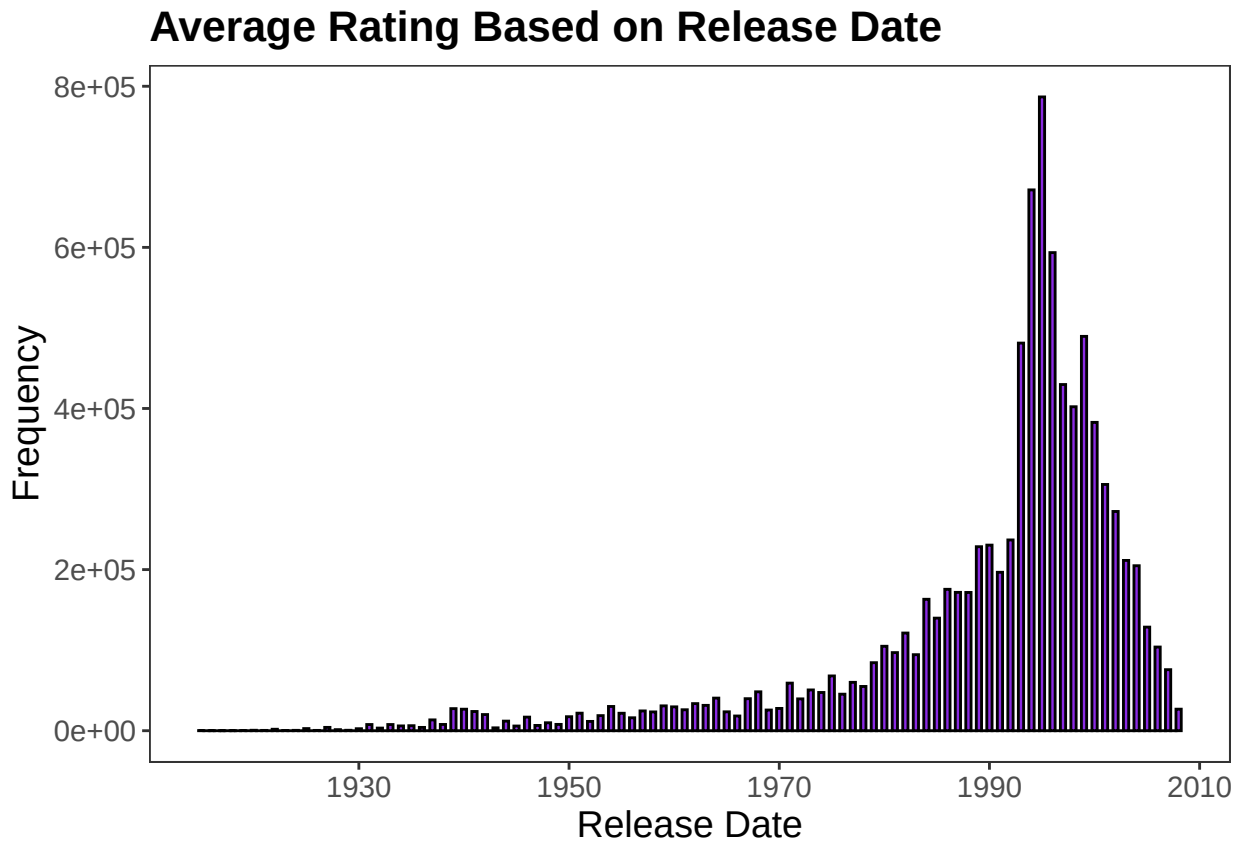
As can be seen, users who have rated movies tend to assign higher scores in general (most ratings are a 4-star rating, following by a 3-star rating). Moreover, as previously mentioned users differ from one another regarding how critical they are, and while some users give higher star rates to majority of movies they rate, others leave lower star ratings. This can be shown via the following code:

```
# Average movie ratings by users
edx %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(binwidth = 0.25, color="black", fill="blueviolet") +
  xlab("Average Star Ratings") +
  ylab("Number of Users") +
  ggtitle("Average Movie Ratings by Users") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) + theme_bw() +
  theme(plot.title = element_text(size = 16, face = "bold"),
        axis.text.y   = element_text(size=11),
        axis.text.x   = element_text(size=11),
        axis.title.y  = element_text(size=14),
        axis.title.x  = element_text(size=14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background=element_rect(fill = "white"))
```



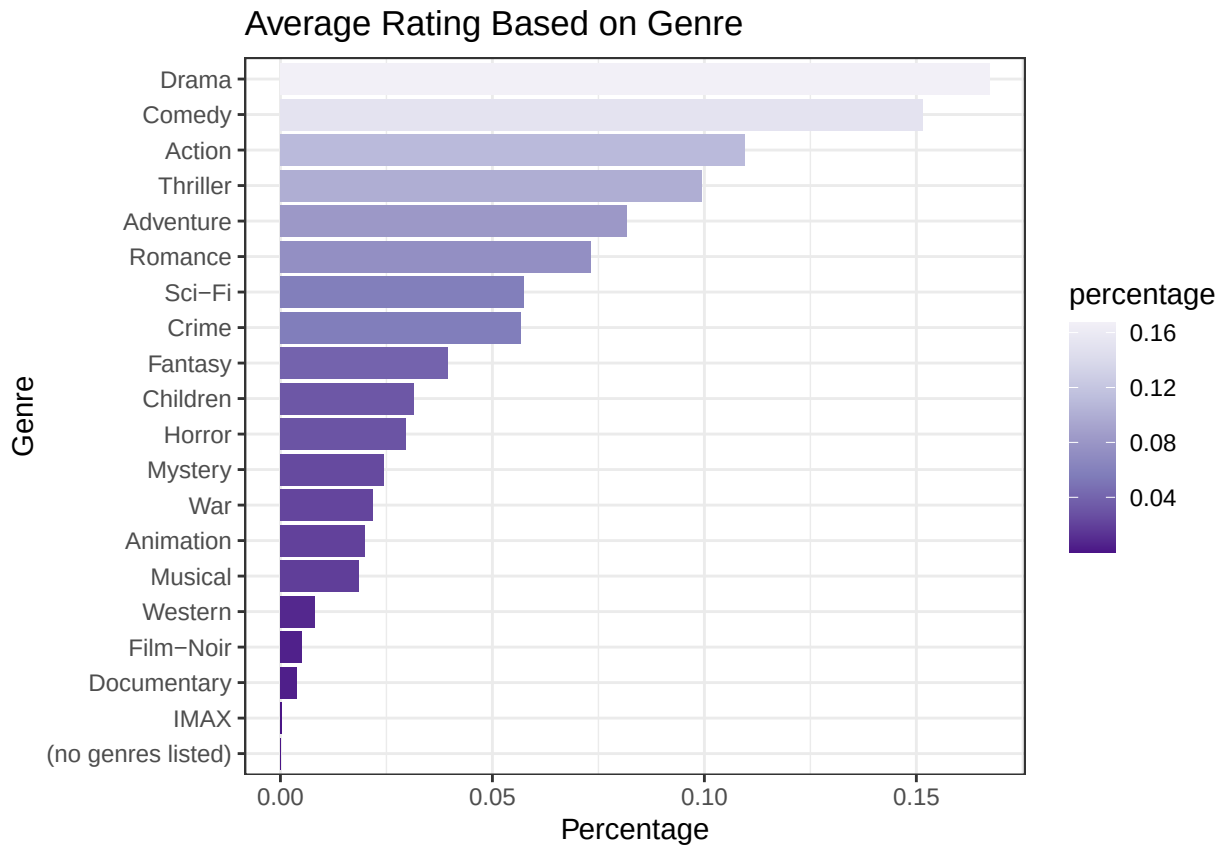
Based on the plots shown above, users and movies effects are apparent. Therefore, it is important to account for these effects in the machine learning models. It is noteworthy that, if one digs more into the data, they could potentially observe some other effects that could be included in the machine learning algorithms. Based upon common sense, for example, that the older the movie, the higher its number of ratings will be. This can be shown in the following plot:

```
# Distribution of rated movies based on the release date
edx$year <- as.numeric(substr(as.character(edx$title),nchar(as.character(edx$title))-4,
                              nchar(as.character(edx$title))-1))

edx %>%
  ggplot(aes(edx$year)) +
  geom_histogram(binwidth = 0.5, color="black", fill="blueviolet") +
  xlab("Release Date") +
  ylab("Frequency") +
  ggtitle("Average Rating Based on Release Date") + theme_bw() +
  theme(plot.title = element_text(size = 16, face = "bold"),
        axis.text.y   = element_text(size=11),
        axis.text.x   = element_text(size=11),
        axis.title.y  = element_text(size=14),
        axis.title.x  = element_text(size=14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background=element_rect(fill = "white"))
```



Similarly and again, based on common sense, different genres can be more popular among users, and thus, movies in certain genres could get rated more frequently than others. In order to see this, we first need to split the movies into single genres. The distribution of rated movies based on genre can then be plotted, using the following code:

```
# Average rating based on genre
dat <- edx %>% separate_rows(genres, sep ="\\|")

dat %>%
  group_by(genres) %>%
  summarize(n=n()) %>%
  ungroup() %>%
  mutate(sumN = sum(n), percentage = n/sumN) %>%
  arrange(-percentage) %>%
  ggplot(aes(reorder(genres, percentage), percentage, fill= percentage)) +
  geom_bar(stat = "identity") + coord_flip() +
  scale_fill_distiller(palette = "Purples") + labs(y = "Percentage", x = "Genre") +
  ggtitle("Average Rating Based on Genre") + theme_bw()
```



Of course, there are other observations that could be made from the available dataset. Note that, however, the focus in this project will be only on the movies and users effects. Nonetheless, in order to investigate the results from a more advanced machine learning algorithm, the recommendation system developed by Chin et al. [10] is used for the sake of comparison. This method is explained as the "Method V: Parallel Matrix Factorization Model" in the next section.

5 Methodology and Procedure

In this project, five different methods are implemented in order to develop the recommendation systems. In all the models,the Mean Square Error (RMSE) has been used as a metric to evaluate the performance of each machine learning (ML) model. The RMSE is a frequently used measure of the differences between values predicted by a model and the values actually observed, and can be defined as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

```
RMSE <- function(predicted_ratings, true_ratings){
  sqrt(mean((predicted_ratings - true_ratings)^2))
}
```

where N is the number of user/movie combinations (i.e. samples), $\hat{y}_{u,i}$ are the predicted values, and $y_{u,i}$ are the observed values. Of course, the lower this metric is, the more accurate the model is.

5.1 Method I: Average Movie Rating

This model is a simple model that predicts the same rating for all movies, and ignores all the variability within the data, explained in the previous section. The mathematical formulation for this model is as follows:

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

with $\epsilon_{u,i}$ being the independent errors sampled from the same distribution centered at 0 and μ the true rating for all movies. This should be apparent that for this specific model, the value that minimizes the RMSE is the average of all ratings:

The average of the all ratings is:

```
mu <- mean(edx$rating)
cat("mu = ", mu)
```

```
## mu =  3.512465
```

If we predict all unknown ratings with μ, we obtain the first naive RMSE:

```
naive_rmse <- RMSE(validation$rating, mu)
cat("RMSE = ", naive_rmse)
```

```
## RMSE =  1.061202
```

In order to keep track of the results, we make a results table and save the RMSE values in this table for the sake of comparison:

```
rmse_results <- data_frame(Method = "Method I: Average Movie Rating", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

Method	RMSE
Method I: Average Movie Rating	1.061202

This RMSE can be regarded as the loss value for the most basic ML model, and can be used as a baseline to evaluate the results from our next ML models.

In the next three models, we will be using some of the insights discussed in the previous section, in order to improve our base ML model.

5.2 Method II: Movie Effect Model

In this model, we will specifically add the movie effects to our base model as follows:

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

where b_i represents average ranking for movie i. As previously discussed, some movies are generally rated higher than others, which could be due to their more popularity among users or the movie age. This effect has been incorporated in our basic model using the term b_i.

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_2_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Method II:  Movie Effect Model",
                                     RMSE = model_2_rmse ))
rmse_results %>% knitr::kable()
```

Method	RMSE
Method I: Average Movie Rating	1.0612018
Method II: Movie Effect Model	0.9439087

In this context, if a movie is on average rated below the average rating for all movies μ, the model predicts that it will be rated lower than μ by b_i (i.e. the difference of the individual movie average from the total average). Therefore, this model can predict the outcomes more accurately.

5.3 Method III: Movie & User Effect Model

In this model, we will add both the movie and user effects to our base model as follows:

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

where b_u is a user-specific effect. As previously mentioned, it makes sense that some users are more cranky while others love every movie they watch. Thus, in this model, if a cranky user (negative b_u) rates a high-ranked movie (positive b_i), the effects counter each other and we may be able to correctly predict that this cranky user may have given this good movie a 3-star instead of 5-star.

Using this model which accounts for both user and movie effects, will result in more accurate predictions, as can be seen below:

```
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings <- validation%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_3_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Method III: Movie & User Effect Model",
```

```
                                    RMSE = model_3_rmse))
rmse_results %>% knitr::kable()
```

Method	RMSE
Method I: Average Movie Rating	1.0612018
Method II: Movie Effect Model	0.9439087
Method III: Movie & User Effect Model	0.8653488

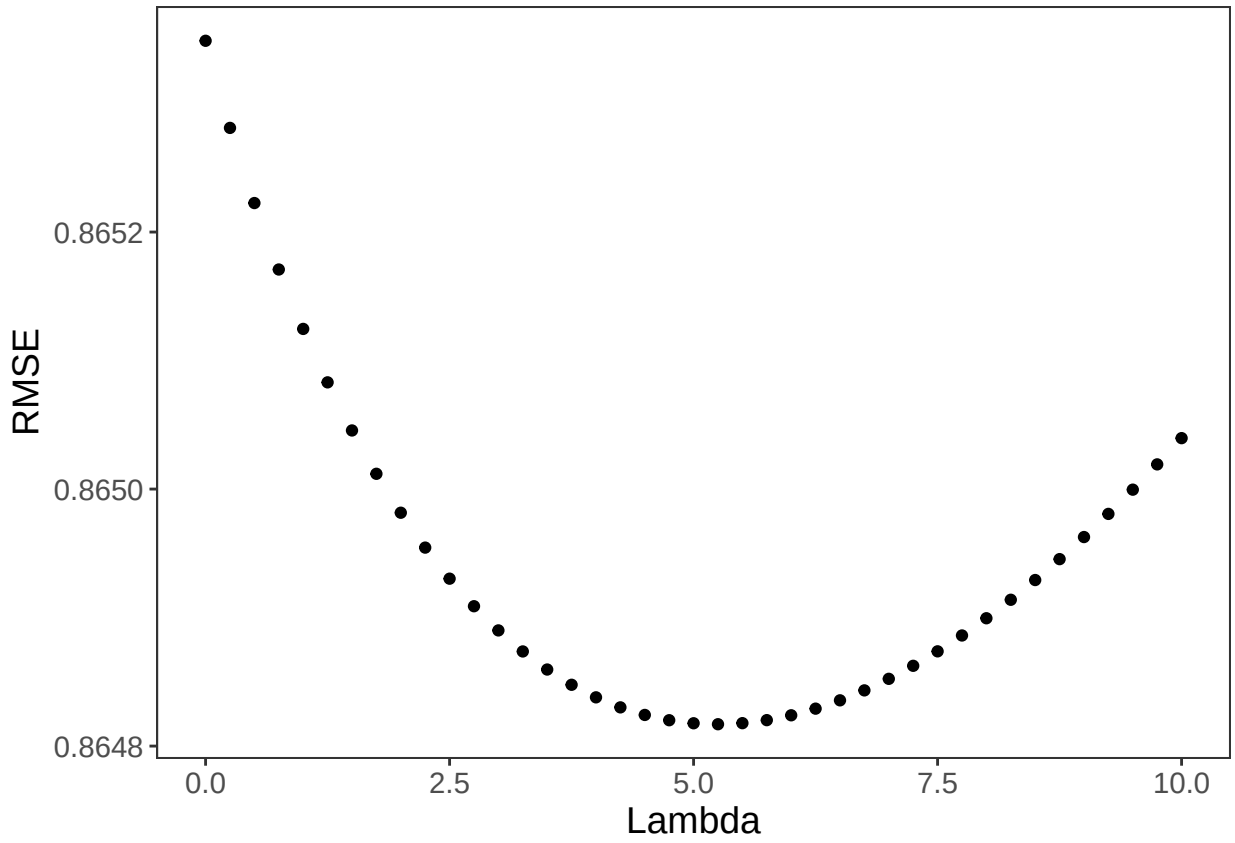
5.4 Method IV: Regularized Movie & User Effect Model

In this model, we use regularization in order to make more accurate predictions. Specifically, model regularization is a fundamental technique for improving the generalization performance of a predictive model. Accordingly, many efficient optimization algorithms have been developed for solving various machine learning formulations with different regularizations [11].

For our specific problem, using a simple data analysis we can learn that the supposed best and worst movies are rated by very few users. Of course, with few users, we have more uncertainty. Thus, larger estimates of b_i are more likely to happen. Regularization permits to penalize large estimates that are formed using small sample sizes. The general idea is to constrain the total variability of the effect sizes. In this context, instead of minimizing the least squares equation for our previous model, we minimize an equation that adds a penalty, as shown below:

$$\Sigma(y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\Sigma b_i^2 + \Sigma b_u^2)$$

where λ is a tuning parameter and can be calculated using cross-validation. The optimal λ for the full model can be calculated as shown below:



The RMSE calculated using this model can be then calculated, using the following:

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Method IV: Regularized Movie & User Effect Model"
                                     , RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

Method	RMSE
Method I: Average Movie Rating	1.0612018
Method II: Movie Effect Model	0.9439087
Method III: Movie & User Effect Model	0.8653488
Method IV: Regularized Movie & User Effect Model	0.8648170

As can be seen, using the concept of reguralization we can build a ML model that can predict the results more accurately, in comparison to the previous model explained.

5.5 Method V: Parallel Matrix Factorization Model

In this model, We will use a fast parallel stochastic gradient method for matrix factorization to predict the outcomes. Using the *recosystem* package in R as the wrapper of the $LIBMF$ library, we can create a recommender system that will predict unknown entries in the rating matrix based on observed values [10].

A popular technique to solve the recommender system problem is the matrix factorization method. The idea is to approximate the whole rating matrix $R_{m \times n}$ by the product of two matrices of lower dimensions, $P_{k \times m}$

and $Q_{k\times n}$, such that:

$$R \approx P'Q$$

Let p_u be the u-th column of P, and q_v be the v-th column of Q, then the rating given by user u on item v would be predicted as $p'_u q_v$.

A typical solution for P and Q is given by the following optimization problem [12-13]:

$$\min_{P,Q} \sum [f(p_u, q_v; r_{u,v}) + \mu P||p_u||_1 + \mu Q||q_v||_1 + \frac{\lambda_P}{2}||p_u||_2^2 + \frac{\lambda_Q}{2}||q_v||_2^2]$$

In this methodology, the data used for training needs to be arranged in sparse matrix triplet form, (i.e. each line in the file should contain three numbers) in order to use *recosystem* package. Moreover, the files will need to be saved as tables. Thus, we will create two new matrices (for training and validation) using the features *movieId*, *userId*, and *rating*, as shown below:

```
edx_factorization <- edx %>% select(movieId, userId, rating)
validation_factorization <- validation %>% select(movieId, userId, rating)

edx_factorization <- as.matrix(edx_factorization)
validation_factorization <- as.matrix(validation_factorization)

write.table(edx_factorization, file = "trainingset.txt", sep = " ", row.names = FALSE,
    col.names = FALSE)

write.table(validation_factorization, file = "validationset.txt", sep = " ",
    row.names = FALSE, col.names = FALSE)

set.seed(1)
training_dataset <- data_file("trainingset.txt")

validation_dataset <- data_file("validationset.txt")

r = Reco() # this will create a model object
```

Similar to the previous model, we tune the algorithm to find the optimal answer:

```
opts = r$tune(training_dataset, opts = list(dim = c(10, 20, 30), lrate = c(0.1,
    0.2), costp_l1 = 0, costq_l1 = 0, nthread = 1, niter = 10))
```

Using the tuned parameters, we can train the model using the training set, as follows:

```
r$train(training_dataset, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9736   1.2042e+07
##    1       0.8721   9.8790e+06
##    2       0.8378   9.1536e+06
##    3       0.8168   8.7473e+06
##    4       0.8012   8.4729e+06
##    5       0.7892   8.2723e+06
##    6       0.7795   8.1204e+06
```

```
##    7       0.7713   7.9959e+06
##    8       0.7646   7.9044e+06
##    9       0.7588   7.8245e+06
##   10       0.7537   7.7575e+06
##   11       0.7492   7.7015e+06
##   12       0.7452   7.6522e+06
##   13       0.7415   7.6098e+06
##   14       0.7382   7.5712e+06
##   15       0.7351   7.5374e+06
##   16       0.7322   7.5068e+06
##   17       0.7296   7.4768e+06
##   18       0.7271   7.4528e+06
##   19       0.7249   7.4299e+06
```

```
stored_prediction = tempfile()
```

We can then use the *predict*() function to make predictions on the validation set, as follows:

```
r$predict(validation_dataset, out_file(stored_prediction))
```

```
## prediction output generated at C:\Users\ADMINI~1\AppData\Local\Temp\2\RtmpqkIIom\filec94569b5d39
```

```
real_ratings <- read.table("validationset.txt", header = FALSE, sep = " ")$V3
pred_ratings <- scan(stored_prediction)

model_5_rmse <- RMSE(real_ratings, pred_ratings)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Method V: Parallel Matrix Factorization Model",
                                     RMSE = model_5_rmse ))
rmse_results %>% knitr::kable()
```

Method	RMSE
Method I: Average Movie Rating	1.0612018
Method II: Movie Effect Model	0.9439087
Method III: Movie & User Effect Model	0.8653488
Method IV: Regularized Movie & User Effect Model	0.8648170
Method V: Parallel Matrix Factorization Model	0.7829341

As can be seen, this ML model provides the most accurate predictions, by minimizing the RMSE in comparison to other models.

6 Results

The RMSE values obtained form the models considered in this project are as follows:

As can be seen, the best model is the Parallel Matrix Factorization Model, which results in a RMSE of 0.78. It is noteworthy that there are other ML models that could be used in order to develop an efficient recommendation system. Examples include Neural Networks and Evolutionary Algorithms. Depending on how the parameters are tuned and what variables are considered as predictors, one may achieve better results

using the aforementioned methodologies. Implementing such models was beyond the scope of this project, however, and thus, the Parallel Matrix Factorization Model is introduced as the most efficient model in this project.

7 Summary and Conclusion

In this project, exploratory data analysis is used in order to develop various machine learning algorithms that predict movie ratings with reasonable accuracy. The project is part of the capstone for the professional certificate in data science program at Harvard University. The MovieLens 10M Dataset [8] that includes 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users is used for creating a movie recommendation system. Considering five different models, the Mean Square Error (RMSE) is used to evaluate how close the predictions are to the true values in the final hold-out test set. The model with the lowest RMSE is reported as the best model. It should be noted that, in the machine learning models, the final hold-out test set (i.e. the validation data) has not been used for training, developing, or selecting the algorithm, and has been only used for evaluating the RMSE of the final algorithm.

Considering the results from the five different models considered in this project, the best model was the Parallel Matrix Factorization Model, which results in a RMSE of 0.78. The model is also reasonably fast, and therefore, is reported as the most efficient algorithm to make the predictions in this project. It should be noted that, however, there are other ML models that could be implemented in order to develop an efficient recommendation system. Examples include Neural Networks and Evolutionary Algorithms. Depending on how the parameters are tuned for each model and what variables are considered as predictors, better predictions might be achieved using the aforementioned methodologies. Implementing such models was beyond the scope of this project, however, and not considered in the project.

8 Acknowledgments

This project is submitted as part of the capstone project for the professional certificate in data science program at Harvard University. The content expressed in this report are the views of the author and do not necessarily represent the opinions or views of the instructors.

9 References

[1] Wang, Z., Yu, X., Feng, N., & Wang, Z. (2014). An improved collaborative movie recommendation system using computational intelligence. Journal of Visual Languages & Computing, 25(6), 667-675.

[2] Katarya, R., & Verma, O. P. (2017). An effective collaborative movie recommender system with cuckoo search. Egyptian Informatics Journal, 18(2), 105-112.

[3] Lu, J., Wu, D., Mao, M., Wang, W., & Zhang, G. (2015). Recommender system application developments: a survey. Decision Support Systems, 74, 12-32.

[4] Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. Knowledge-based systems, 46, 109-132.

[5] Chen, L., Chen, G., & Wang, F. (2015). Recommender systems based on user reviews: the state of the art. User Modeling and User-Adapted Interaction, 25(2), 99-154.

[6] Konstan, J. A., & Riedl, J. (2012). Recommender systems: from algorithms to user experience. User modeling and user-adapted interaction, 22(1-2), 101-123.

[7] Katarya, R., & Verma, O. P. (2016). Recent developments in affective recommender systems. Physica A: Statistical Mechanics and its Applications, 461, 182-190.

[8] https://grouplens.org/datasets/movielens/10m/

[9] https://grouplens.org/

[10] Chin, W. S., Yuan, B. W., Yang, M. Y., Zhuang, Y., Juan, Y. C., & Lin, C. J. (2016). LIBMF: a library for parallel matrix factorization in shared-memory systems. The Journal of Machine Learning Research, 17(1), 2971-2975.

[11] Zhu, D., Cai, C., Yang, T., & Zhou, X. (2018). A machine learning approach for air quality prediction: Model regularization and optimization. Big data and cognitive computing, 2(1), 5.

[12] Chin, W. S., Zhuang, Y., Juan, Y. C., & Lin, C. J. (2015). A fast parallel stochastic gradient method for matrix factorization in shared memory systems. ACM Transactions on Intelligent Systems and Technology (TIST), 6(1), 1-24.

[13] Chin, W. S., Zhuang, Y., Juan, Y. C., & Lin, C. J. (2015, May). A learning-rate schedule for stochastic gradient methods to matrix factorization. In Pacific-Asia Conference on Knowledge Discovery and Data Mining (pp. 442-455). Springer, Cham.