Movie Recommendation System in R

Nirmal Janapaneedi

04/23/2020

Contents

Preface

This report is part of the capstone project of HarvardX's Data Science Professional Certificate[1] program.

1 https://www.edx.org/professionalcertificate/harvardxdatascience

Chapter 1

Introduction

Recommendation systems plays an important role in ecommerce and online streaming services, such as Netflix, YouTube and Amazon. Making the right recommendation for the next product, music or movie in creases user retention and satisfaction, leading to sales and profit growth. Companies competing for cus tomer loyalty invest on systems that capture and analyses the user's preferences, and offer products or services with higher likelihood of purchase.

The economic impact of such companycustomer relationship is clear: Amazon is the largest online retail company by sales and part of its success comes from the recommendation system and marketing based on user preferences. In 2006 Netflix offered a one million dollar prize[1] for the person or group that could improve their recommendation system by at least 10%.

Usually recommendation systems are based on a rating scale from 1 to 5 grades or stars, with 1 indicating lowest satisfaction and 5 is the highest satisfaction. Other indicators can also be used, such as comments posted on previously used items; video, music or link shared with friends; percentage of movie watched or music listened; web pages visited and time spent on each page; product category; and any other interaction with the company's web site or application can be used as a predictor.

The primary goal of recommendation systems is to help users find what they want based on their preferences and previous interactions, and predicting the rating for a new item. In this document, we create a movie rec ommendation system using the MovieLens dataset and applying the lessons learned during the HarvardX's Data Science Professional Certificate[2] program.

This document is structured as follows. Chapter 1 describes the dataset and summarizes the goal of the project and key steps that were performed. In chapter 2 we explain the process and techniques used, such as data cleaning, data exploration and visualization, any insights gained, and the modeling approach. In chapter 3 we present the modeling results and discuss the model performance. We conclude in chapter 4 with a brief summary of the report, its limitations and future work.

1 https://www.netflixprize.com/

2 https://www.edx.org/professionalcertificate/harvardxdatascience

1.1 MovieLens Dataset

GroupLens[3] is a research lab in the University of Minnesota that has collected and made available rating data for movies in the MovieLens web site[4].

The complete MovieLens dataset[5] consists of 27 million ratings of 58,000 movies by 280,000 users. The research presented in this paper is based in a subset[6] of this dataset with 10 million ratings on 10,000 movies by 72,000 users.

1.2 Model Evaluation

The evaluation of machine learning algorithms consists in comparing the predicted value with the actual outcome. The loss function measures the difference between both values. There are other metrics that go beyond the scope of this document.

The most common loss functions in machine learning are the mean absolute error (MAE), mean squared error (MSE) and root mean squared error (RMSE).

Regardless of the loss function, when the user consistently selects the predicted movie, the error is equal to zero and the algorithm is perfect.

The goal of this project is to create a recommendation system with RMSE lower than 0.8649. There's no specific target for MSE and MAE.

1.2.1 Mean Absolute Error MAE

The mean absolute error is the average of absolute differences between the predicted value and the true value. The metric is linear, which means that all errors are equally weighted. Thus, when predicting ratings in a scale of 1 to 5, the MAE assumes that an error of 2 is twice as bad as an error of 1.

The MAE is given by this formula:

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i|$$

where N is the number of observations, $\hat{y}_i$ is the predicted value and y is the true value.

1.2.2 Mean Squared Error MSE

The MSE is the average squared error of the predictions. Larger errors are weighted more than smaller ones, thus if the error doubles, the MSE increases four times. On the other hand, errors smaller than 1 also decrease faster, for example an error of 0.5 results in MSE of 0.25.

[3] https://grouplens.org/

[4] https://movielens.org/

[5] https://grouplens.org/datasets/movielens/latest/

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

$$MSE = \frac{1}{N} \sum_{u,i} (\hat{y}_i - y_i)^2$$

1.2.3 Root Mean Squared Error RMSE

The Root Mean Squared Error, RMSE, is the square root of the MSE. It is the typical metric to evaluate recommendation systems, and is defined by the formula:

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

where N is the number of ratings, $y_{u,i}$ is the rating of movie i by user u and $\hat{y}_{u,i}$ is the prediction of movie i by user u.

Similar to MSE, the RMSE penalizes large deviations from the mean and is appropriate in cases that small errors are not relevant. Contrary to the MSE, the error has the same unit as the measurement.

1.3 Process and Workflow

The main steps in a data science project include:

1. Data preparation: download, parse, import and prepare the data to be processed and analysed.
2. Data exploration and visualization: explore data to understand the features and the relationship be tween the features and predictors.
3. Data cleaning: eventually the dataset contains unnecessary information that needs to be removed.
4. Data analysis and modeling: create the model using the insights gained during exploration. Also test and validate the model.
5. Communicate: create the report and publish the results.

First we download the dataset from MovieLens website and split into two subsets used for training and validation. The training subset is called `edx` and the validation subset is called `validation`. The `edx` set is split again into two subsets used for training and and testing. When the model reaches the RMSE target in the testing set, we train the `edx` set with the model and use the `validation` set for final validation. We pretend the `validation` set is new data with unknown outcomes.

In the next step we create charts, tables and statistics summary to understand how the features can impact the outcome. The information and insights obtained during exploration will help to build the machine learning model.

Creating a recommendation system involves the identification of the most important features that helps to predict the rating any given user will give to any movie. We start building a very simple model, which is just the mean of the observed values. Then, the user and movie effects are included in the linear model, improving

the RMSE. Finally, the user and movie effects receive regularization parameter that penalizes samples with few ratings.

Although the linear model with regularization achieves the desired RMSE, matrix factorization using the LIBMF[7] algorithm is evaluated and provides a better prediction. LIBMF is available through the R package `recosystem`[8].

[7] A Matrixfactorization Library for Recommender Systems https://www.csie.ntu.edu.tw/~cjlin/libmf/

[8] https://cran.rproject.org/web/packages/recosystem/index.html

Chapter 2

Methods and Analysis

2.1 Data Preparation

In this section we download and prepare the dataset to be used in the analysis. We split the dataset in two parts, the training set called `edx` and the evaluation set called `validation` with 90% and 10% of the original dataset respectively.

Then, we split the `edx` set in two parts, the `train` set and `test` set with 90% and 10% of `edx` set respec tively. The model is created and trained in the `train` set and tested in the `test` set until the RMSE target is achieved, then finally we train the model again in the entire `edx` set and validate in the `validation` set. The name of this method is *crossvalidation*.

```
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")

# 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)
```

```r
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(levels(movieId))[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")
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)
```

The `edx` set is used for training and testing, and the `validation` set is used for final validation to simulate the new data.

Here, we split the `edx` set in 2 parts: the training set and the test set.

The model building is done in the training set, and the test set is used to test the model. When the model is complete, we use the `validation` set to calculate the final RMSE.

We use the same procedure used to create `edx` and `validation` sets.

The training set will be 90% of `edx` data and the test set will be the remaining 10%.

```r
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in test set are also in
train set test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>
  % semi_join(train_set, by = "userId")
```

```
# Add rows removed from test set back into
train set removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

2.2 Data Exploration

Before start building the model, we need to understand the structure of the data, the distribution of ratings and the relationship of the predictors. This information will help build a better model.

```
str(edx)
```

```
 ## 'data.frame':   9000055 obs. of 6 variables:
 ##  $ userId   : int  1 1 1 1111      111...
#   $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
## $ rating  : num 5 5 5 5 5 5 5 5 5 ...
#   $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983
## $ title   : chr "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (
## $ genres  : chr "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thri
```

From this initial exploration, we discover that `edx` has 6 columns:

movieId	integer
userId	integer
rating	numeric
timestamp	numeric
title	character
genres	character

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|SciFi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|SciFi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|SciFi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

The next sections discover more details about each feature and outcome.

2.2.1 Genres

Along with the movie title, MovieLens provides the list of genres for each movie. Although this information can be used to make better predictions, this research doesn't use it. However it's worth exploring this information as well.
The data set contains 797 different combinations of genres. Here is the list of the first six.

```
edx %>% group_by(genres) %>%
  summarise(n=n()) %>%
  head()
```

genres	n
(no genres listed)	7
Action	24482
Action|Adventure	68688
Action|Adventure|Animation|Children|Comedy	7467
Action|Adventure|Animation|Children|Comedy|Fantasy	187
Action|Adventure|Animation|Children|Comedy|IMAX	66

The table above shows that several movies are classified in more than one genre. The number of genres in each movie is listed in this table, sorted in descend order.

```
tibble(count = str_count(edx$genres, fixed("|")), genres =
  edx$genres) %>% group_by(count, genres) %>%
  summarise(n = n()) %>%
```

2.2.2 Date

The rating period was collected over almost 14 years.

```
library(lubridate)
tibble(`Initial Date` = date(as_datetime(min(edx$timestamp), origin="1970-01-01")),
       `Final Date` = date(as_datetime(max(edx$timestamp), origin="1970-01-01"))) %>%
  mutate(Period = duration(max(edx$timestamp)-min(edx$timestamp)))
```

Initial Date	Final Date	Period
19950109	20090105	441479727s (~13.99 years)

```
if(!require(ggthemes))
  install.packages("ggthemes", repos = "http://cran.us.r-project.org")
if(!require(scales))
  install.packages("scales", repos = "http://cran.us.r-project.org")
edx %>% mutate(year = year(as_datetime(timestamp, origin="1970-
  01-01"))) %>% ggplot(aes(x=year)) +
    geom_histogram(color = "white") +
    ggtitle("Rating Distribution Per Year") +
    xlab("Year") +
    ylab("Number of Ratings") +
    scale_y_continuous(labels = comma) +
    theme_economist()
```

The following table lists the days with more ratings. Not surprisingly, the movies are well known blockbusters.

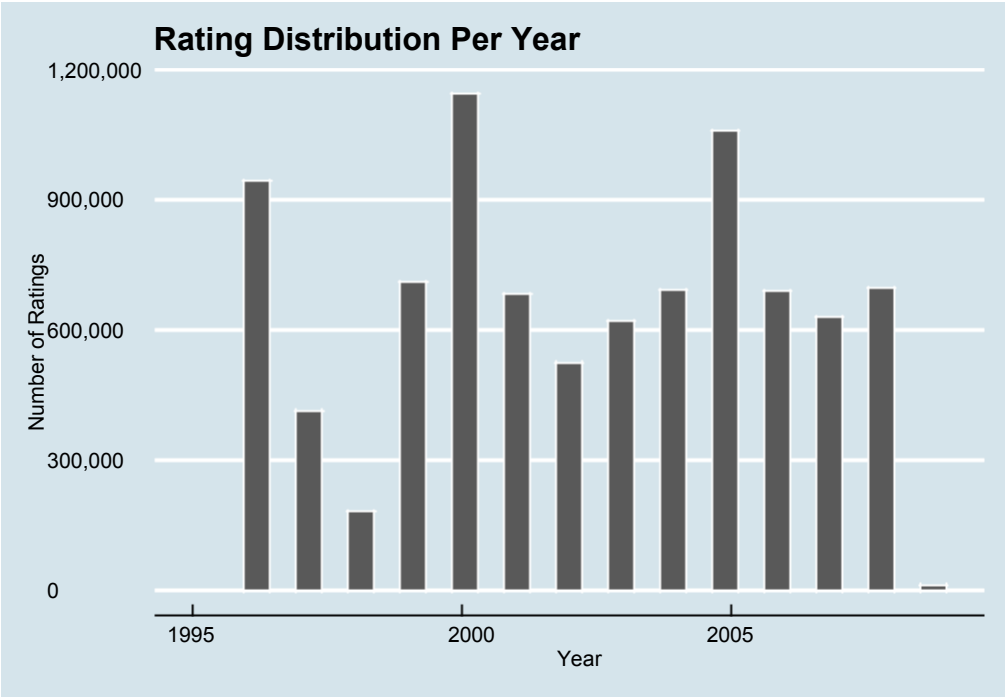


Figure 2.1: Rating distribution per year.

```
edx %>% mutate(date = date(as_datetime(timestamp, origin="1970-
  01-01"))) %>% group_by(date, title) %>%
  summarise(count = n()) %>%
  arrange(-count) %>%
  head(10)
```

date	title	count
19980522	Chasing Amy (1997)	322
20001120	American Beauty (1999)	277
19991211	Star Wars: Episode IV A New Hope (a.k.a. Star Wars) (1977)	254
19991211	Star Wars: Episode V The Empire Strikes Back (1980)	251
19991211	Star Wars: Episode VI Return of the Jedi (1983)	241
20050322	Lord of the Rings: The Two Towers, The (2002)	239
20050322	Lord of the Rings: The Fellowship of the Ring, The (2001)	227
20001120	Terminator 2: Judgment Day (1991)	221
19991211	Matrix, The (1999)	210
20001120	Jurassic Park (1993)	201

2.2.3 Ratings

Users have the option to choose a rating value from 0.5 to 5.0, totaling 10 possible values. This is unusual scale, so most movies get a rounded value rating, as shown in the chart below.

Count the number of each ratings: 13

```
edx %>% group_by(rating) %>% summarize(n=n())
```

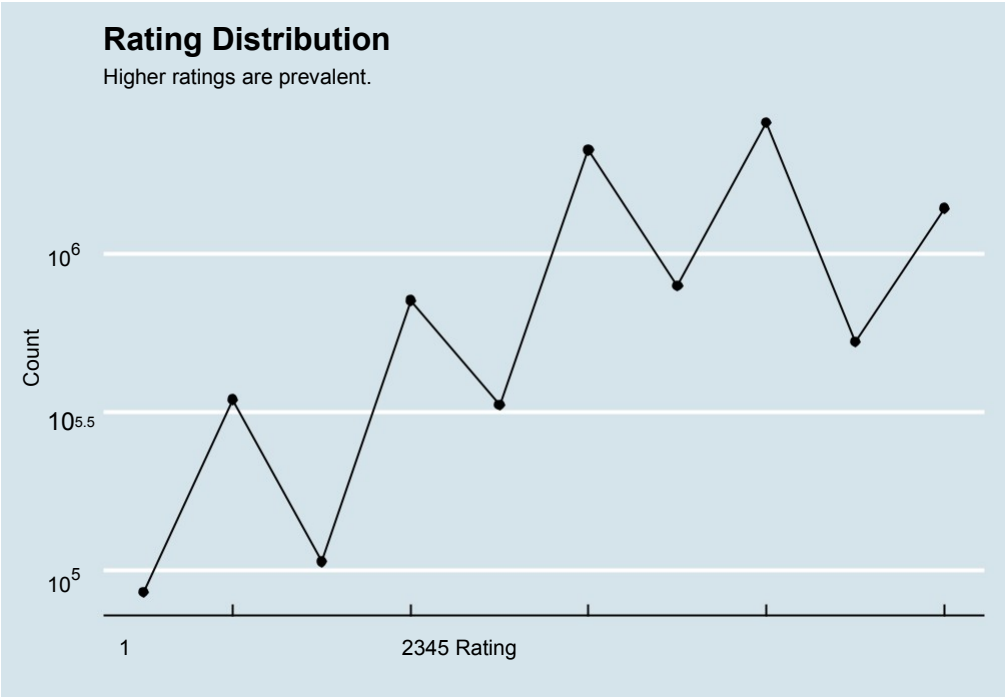


Figure 2.2: Round values receive more ratings than decimals.

```
edx %>% group_by(rating) %>%
  summarise(count=n()) %>%
  ggplot(aes(x=rating, y=count)) +
    geom_line() +
    geom_point() +
    scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                  labels = trans_format("log10", math_format(10^.x))) +
    ggtitle("Rating Distribution", subtitle = "Higher ratings are prevalent.") +
    xlab("Rating") +
    ylab("Count") +
    theme_economist()
```

2.2.4 Movies

There are 10677 different movies in the `edx` set. We know from intuition that some of them are rated more than others, since many movies are watched by few users and blockbusters tend to have more ratings.

```
edx %>% group_by(movieId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
    geom_histogram(color = "white") +
```

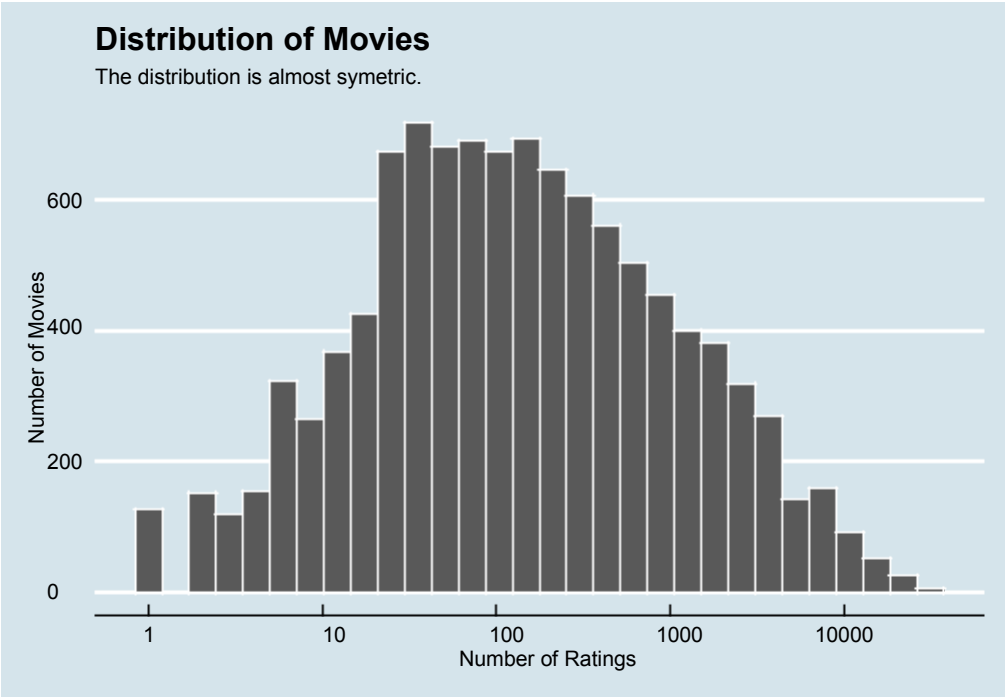


Figure 2.3: 10.7% of movies have less than 10 ratings.

```
    scale_x_log10() +
    ggtitle("Distribution of Movies",
            subtitle = "The distribution is almost symetric.") +
    xlab("Number of Ratings") +
    ylab("Number of Movies") +
    theme_economist()
```

2.2.5 Users

There are 69878 different users are in the `edx` set.

The majority of users rate few movies, while a few users rate more than a thousand movies.

5% users rated less than 20 movies.

```
edx %>% group_by(userId) %>%
  summarise(n=n()) %>%
  arrange(n) %>%
  head()
```

userId	n
62516	10

userId	n
22170	12
15719	13
50608	13
901	14
1833	14

The user distribution is right skewed.

```
edx %>% group_by(userId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
    geom_histogram(color = "white") +
    scale_x_log10() +
    ggtitle("Distribution of Users",
            subtitle="The distribution is right skewed.") +
    xlab("Number of Ratings") +
    ylab("Number of Users") +
    scale_y_continuous(labels = comma) +
    theme_economist()
```

Show the heatmap of users x movies

This usermovie matrix is sparse, with the vast majority of empty cells. Notice that four movies have more ratings, and one or two users are more active.

```
users <- sample(unique(edx$userId), 100)
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
title("User x Movie Matrix")
```

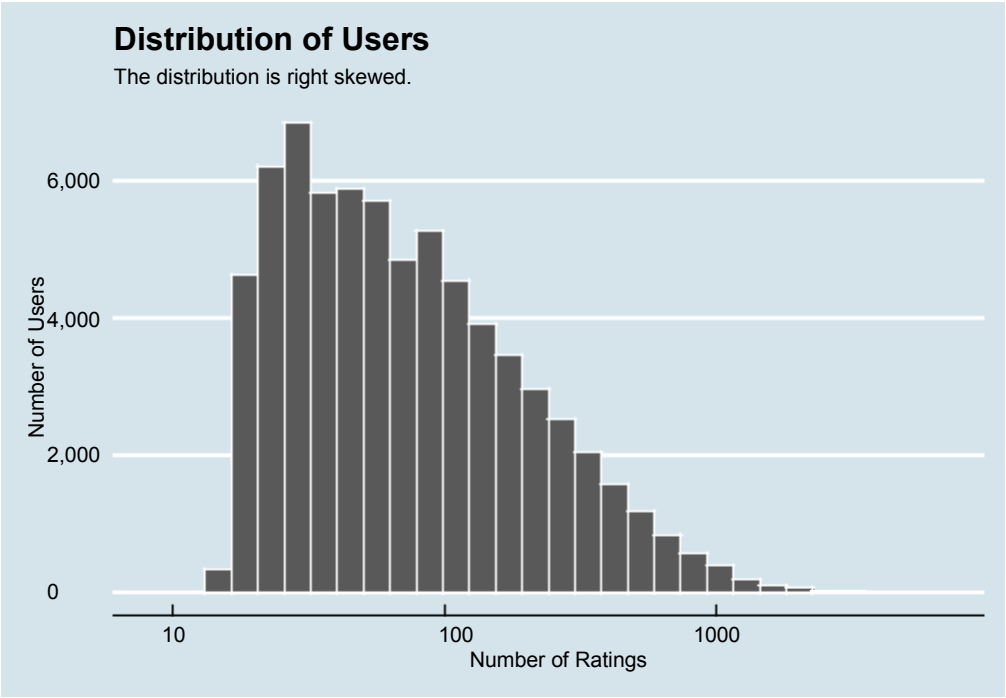
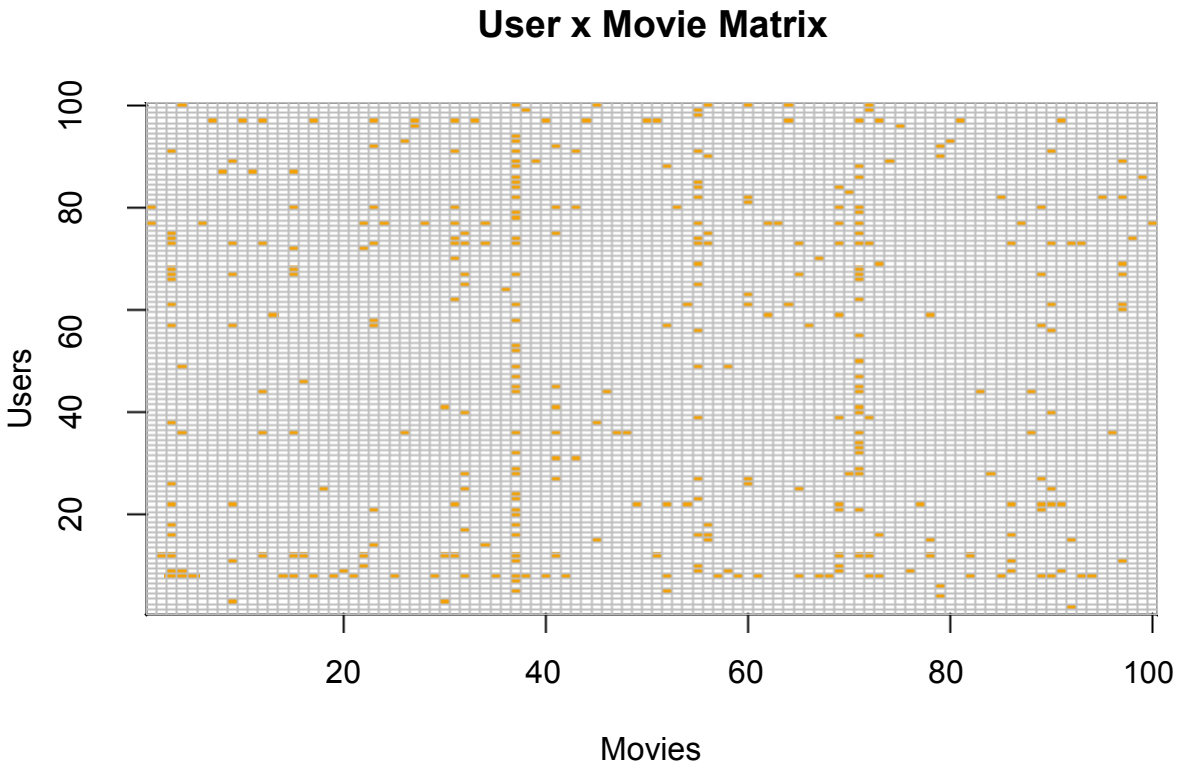


Figure 2.4: 5% of users rated less than 20 movies.



2.3 Data Cleaning

As previously discussed, several features can be used to predict the rating for a given user. However, many predictors increases the model complexity and requires more computer resources, so in this research the estimated rating uses only movie and user information.

```
train_set <- train_set %>% select(userId, movieId, rating, title)
test_set <- test_set %>% select(userId, movieId, rating, title)
```

2.4 Modeling

2.4.1 Random Prediction

A very simple model is just randomly predict the rating using the probability distribution observed during the data exploration. For example, if we know the probability of all users giving a movie a rating of 3 is 10%, then we may guess that 10% of the ratings will have a rating of 3.

Such prediction sets the worst error we may get, so any other model should provide better result.

2.4.2 Linear Model

The simplest model predicts all users will give the same rating to all movies and assumes the movie to movie variation is the randomly distributed error. Although the predicted rating can be any value, statistics theory says that the average minimizes the RMSE, so the initial prediction is just the average of all observed ratings, as described in this formula:

$$\hat{Y}_{u,i} = \mu + \epsilon_{i,u}$$

Where $\hat{Y}$ is the predicted rating, μ is the mean of observed data and $\epsilon_{i,u}$ is the error distribution. Any value other than the mean increases the RMSE, so this is a good initial estimation.

Part of the movie to movie variability can be explained by the fact that different movies have different rating distribution. This is easy to understand, since some movies are more popular than others and the public preference varies. This is called movie effect or movie bias, and is expressed as b_i in this formula:

$$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{i,u}$$

The movie effect can be calculated as the mean of the difference between the observed rating y and the mean μ.

$$\hat{b}_i = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{\mu})$$

Similar to the movie effect, different users have different rating pattern or distribution. For example, some users like most movies and consistently rate 4 or 5, while other users dislike most movies rating 1 or 2. This is called user effect or user bias and is expressed in this formula:

$$\hat{b}_u = \frac{1}{N}\sum_{i=1}^{N}(y_{u,i} - \hat{b}_i - \hat{\mu})$$

The prediction model that includes the user effect becomes:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

Movies can be grouped into categories or genres, with different distributions. In general, movies in the same genre get similar ratings. In this project we won't evaluate the genre effect.

2.4.3 Regularization

The linear model provides a good estimation for the ratings, but doesn't consider that many movies have very few number of ratings, and some users rate very few movies. This means that the sample size is very small for these movies and these users. Statistically, this leads to large estimated error.

The estimated value can be improved adding a factor that penalizes small sample sizes and have have little or no impact otherwise. Thus, estimated movie and user effects can be calculated with these formulas:

$$\hat{b}_i = \frac{1}{n_i + \lambda}\sum_{u=1}^{n_i}(y_{u,i} - \hat{\mu})$$

$$\hat{b}_u = \frac{1}{n_u + \lambda}\sum_{i=1}^{n_u}(y_{u,i} - \hat{b}_i - \hat{\mu})$$

For values of N smaller than or similar to λ, b_i and b_u $\hat{b}_i$ is smaller than the original values, whereas for values of N much larger than λ, b_i and b_u change very little.

An effective method to choose λ that minimizes the RMSE is running simulations with several values of λ.

2.4.4 Matrix Factorization

Matrix factorization is widely used machine learning tool for predicting ratings in recommendation systems. This method became widely known during the Netflix Prize challenge[1].

The data can be converted into a matrix such that each user is in a row, each movie is in a column and the rating is in the cell, then the algorithm attempts to fill in the missing values. The table below provides a simple example of a 4×5 matrix.

1 https://www.netflixprize.com/

	movie 1	movie 2	movie 3	movie 4	movie 5
user 1	?	?	4	?	3
user 2	2	?	?	4	?
user 3	?	3	?	?	5
user 4	3	?	2	?	?

The concept is to approximate a large rating matrix $R_{m \times n}$ into the product of two lower dimension matrices $P_{k \times m}$ and $Q_{k \times n}$, such that

$$R \approx P^{'} Q$$

The R `recosystem`[2] package provides methods to decompose the rating matrix and estimate the user rating, using parallel matrix factorization.

2 https://cran.rproject.org/web/packages/recosystem/vignettes/introduction.html

Chapter 3

Results

This section presents the code and results of the models.

3.1 Model Evaluation Functions

Here we define the loss functions.

```
# Define Mean Absolute Error (MAE)
MAE <- function(true_ratings, predicted_ratings){
  mean(abs(true_ratings - predicted_ratings))
}

# Define Mean Squared Error (MAE)
MSE <- function(true_ratings, predicted_ratings){
  mean((true_ratings - predicted_ratings)^2)
}

# Define Root Mean Squared Error (RMSE)
 RMSE <- function(true_ratings, predicted_ratings){
    sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

3.2 Random Prediction

The first model randomly predicts the ratings using the observed probabilities in the training set. First, we calculate the probability of each rating in the training set, then we predict the rating for the test set and compare with actual rating. Any model should be better than this one.

Since the training set is a sample of the entire population and we don't know the real distribution of ratings, the Monte Carlo simulation with replacement provides a good approximation of the rating distribution.

```r
set.seed(4321, sample.kind = "Rounding")

# Create the probability of each
rating p <- function(x, y) mean(y
== x) rating <- seq(0.5,5,0.5)

# Estimate the probability of each rating with Monte Carlo
simulation B <- 10^3
M <- replicate(B, {
  s <- sample(train_set$rating, 100, replace
  = TRUE) sapply(rating, p, y= s)
})
prob <- sapply(1:nrow(M), function(x) mean(M[x,]))

# Predict random ratings
y_hat_random <- sample(rating, size = nrow(test_set),
                       replace = TRUE, prob = prob)

# Create a table with the error results
result <- tibble(Method = "Project Goal", RMSE = 0.8649, MSE =
NA, MAE = NA) result <- bind_rows(result,
                tibble(Method = "Random prediction",
                         RMSE = RMSE(test_set$rating, y_hat_random),
                         MSE = MSE(test_set$rating, y_hat_random),
                         MAE = MAE(test_set$rating, y_hat_random)))
```

The RMSE of random prediction is very high.

```r
result
```

Method	RMSE	MSE	MAE
Project Goal	0.864900	NA	NA
Random prediction	1.501245	2.253735	1.167597

3.3 Linear Model

We're building the linear model based on the formula:

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

3.3.1 Initial Prediction

The initial prediction is just the mean of the ratings, μ.

$$\hat{y} = \mu + \epsilon_{u,i}$$

```
# Mean of observed values mu
<- mean(train_set$rating)

# Update the error table
result <- bind_rows(result,
                    tibble(Method = "Mean",
                           RMSE = RMSE(test_set$rating, mu),
                           MSE  = MSE(test_set$rating, mu),
                           MAE  = MAE(test_set$rating, mu)))
# Show the RMSE
improvement result
```

Method	RMSE	MSE	MAE
Project Goal	0.864900	NA	NA
Random prediction	1.501245	2.253735	1.1675974
Mean	1.060054	1.123714	0.8551636

3.3.2 Include Movie Effect (b_i)

b_i is the movie effect (bias) for movie i.

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

```
# Movie effects (bi)
bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating -
mu)) head(bi)
```

movieId	b_i
1	0.4150040
2	0.3064057
3	0.3613952
4	0.6372808
5	0.4416058
6	0.3018943

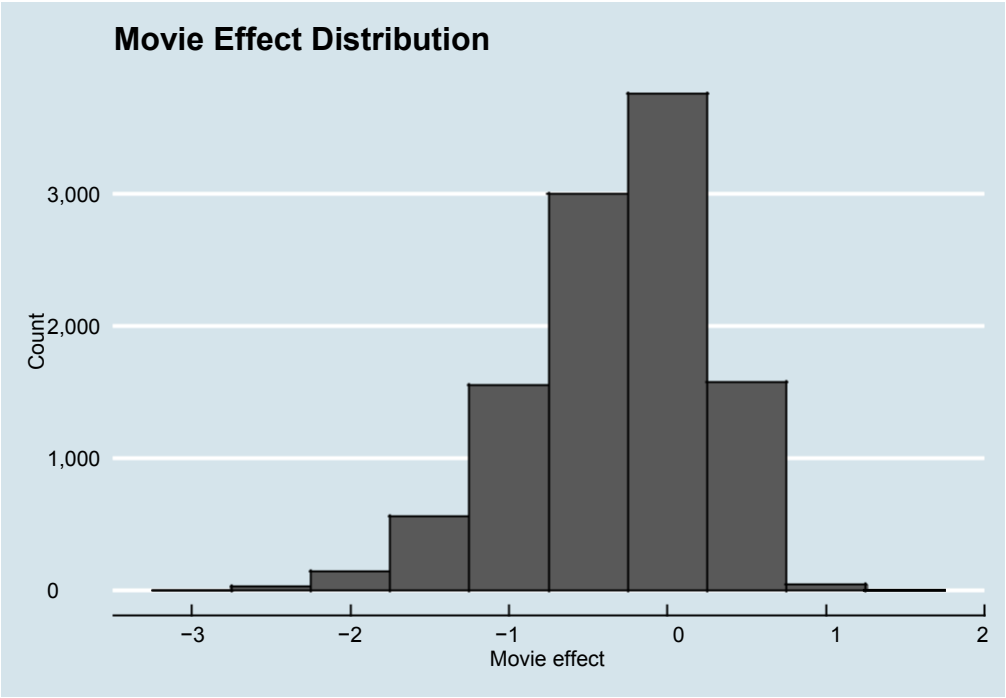


Figure 3.1: Distribution of movie effects.

The movie effect is normally left skewed distributed.

```
bi %>% ggplot(aes(x = b_i)) +
  geom_histogram(bins=10, col = I("black")) +
  ggtitle("Movie Effect Distribution") +
  xlab("Movie effect") +
  ylab("Count") +
  scale_y_continuous(labels = comma) +
  theme_economist()
```

```
# Predict the rating with mean +
bi y_hat_bi <- mu + test_set %>%
  left_join(bi, by = "movieId") %>%
  .$b_i

# Calculate the RMSE
result <- bind_rows(result,
                    tibble(Method = "Mean + bi",
                           RMSE = RMSE(test_set$rating, y_hat_bi),
                           MSE = MSE(test_set$rating, y_hat_bi),
                           MAE = MAE(test_set$rating, y_hat_bi)))
```

```
# Show the RMSE improvement
result
```

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5012445	2.2537350	1.1675974
Mean	1.0600537	1.1237139	0.8551636
Mean + bi	0.9429615	0.8891764	0.7370384

3.3.3 Include User Effect (b_u)

b_u is the user effect (bias) for user u.

$$\hat{y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

Predict the rating with mean + bi + bu

```
# User effect (bu)
bu <- train_set %>%
  left_join(bi, by = 'movieId')
  %>% group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Prediction
y_hat_bi_bu <- test_set %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

# Update the results table
result <- bind_rows(result,
                    tibble(Method = "Mean + bi + bu",
                           RMSE = RMSE(test_set$rating, y_hat_bi_bu),
                           MSE = MSE(test_set$rating, y_hat_bi_bu),
                           MAE = MAE(test_set$rating, y_hat_bi_bu)))

# Show the RMSE
improvement result
```

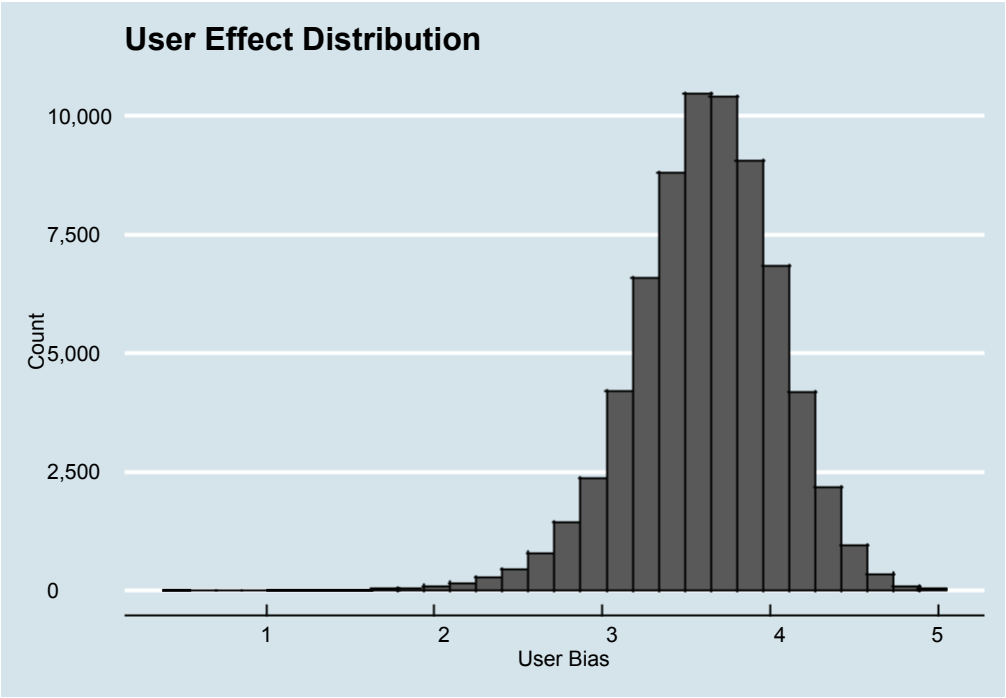


Figure 3.2: User effect is normally distributed.

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5012445	2.2537350	1.1675974
Mean	1.0600537	1.1237139	0.8551636
Mean + bi	0.9429615	0.8891764	0.7370384
Mean + bi + bu	0.8646843	0.7476789	0.6684369

The user effect is normally distributed.

```
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
    geom_histogram(color = "black") +
    ggtitle("User Effect Distribution") +
    xlab("User Bias") +
    ylab("Count") +
    scale_y_continuous(labels = comma) +
    theme_economist()
```

3.3.4 Evaluating the model result

The RMSE improved from the initial estimation based on the mean. However, we still need to check if the model makes good ratings predictions.

Check the 10 largest residual differences

```
train_set %>%
  left_join(bi, by='movieId') %>%
  mutate(residual = rating - (mu + b_i))
  %>% arrange(desc(abs(residual))) %>%
  slice(1:10)
```

userId	movieId	rating	title	b_i	residual
26423	6483	5.0	From Justin to Kelly (2003)	2.638139	4.125683
5279	6371	5.0	PokÃ©mon Heroes (2003)	2.472133	3.959677
57863	6371	5.0	PokÃ©mon Heroes (2003)	2.472133	3.959677
2507	318	0.5	Shawshank Redemption, The (1994)	0.944111	3.956567
7708	318	0.5	Shawshank Redemption, The (1994)	0.944111	3.956567
9214	318	0.5	Shawshank Redemption, The (1994)	0.944111	3.956567
9568	318	0.5	Shawshank Redemption, The (1994)	0.944111	3.956567
9975	318	0.5	Shawshank Redemption, The (1994)	0.944111	3.956567
10749	318	0.5	Shawshank Redemption, The (1994)	0.944111	3.956567
13496	318	0.5	Shawshank Redemption, The (1994)	0.944111	3.956567

```
titles <- train_set %>%
  select(movieId, title) %>%
  distinct()
```

Top 10 best movies (ranked by b_i).
These are unknown movies

```
bi %>%
  inner_join(titles, by = "movieId") %>%
  arrange(-b_i) %>%
  select(title) %>%
  head()
```

title
Hellhounds on My Trail (1999)
Satan’s Tango (SÃ¡tÃ¡ntangÃ³) (1994)
Shadows of Forgotten Ancestors (1964)

title
Fighting Elegy (Kenka erejii) (1966)
Sun Alley (Sonnenallee) (1999)
Blue Light, The (Das Blaue Licht) (1932)

Top 10 worst movies (ranked by b_i), also unknown movies:

```
bi %>%
  inner_join(titles, by = "movieId") %>%
  arrange(b_i) %>%
  select(title) %>%
  head()
```

title
Besotted (2001)
HiLine, The (1999)
Accused (Anklaget) (2005)
Confessions of a Superhero (2007)
War of the Worlds 2: The Next Wave (2008)
SuperBabies: Baby Geniuses 2 (2004)

Number of ratings for 10 best movies:

```
train_set %>%
  left_join(bi, by = "movieId") %>%
  arrange(desc(b_i)) %>%
  group_by(title) %>%
  summarise(n = n()) %>%
  slice(1:10)
```

title	n
'burbs, The (1989)	1201
'night Mother (1986)	178
'Round Midnight (1986)	40
'Til There Was You (1997)	242
"Great Performances" Cats (1998)	4
batteries not included (1987)	389
...All the Marbles (a.k.a. The California Dolls) (1981)	17
...And God Created Woman (Et Dieu... crÃ©a la femme) (1956)	68
...And God Spoke (1993)	19
...And Justice for All (1979)	500

```
train_set %>% count(movieId) %>%
  left_join(bi, by="movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(n)
```

```
#   [1]1111114244
```

3.4 Regularization

Now, we regularize the user and movie effects adding a penalty factor λ, which is a tuning parameter. We define a number of values for λ and use the `regularization` function to pick the best value that minimizes the RMSE.

```
regularization <- function(lambda, trainset, testset){

  # Mean
  mu <- mean(trainset$rating)

  # Movie effect (bi)
  b_i <- trainset %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

  # User effect (bu)
  b_u <- trainset %>%
    left_join(b_i, by="movieId") %>%
    filter(!is.na(b_i)) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

  # Prediction: mu + bi + bu
  predicted_ratings <- testset %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    filter(!is.na(b_i), !is.na(b_u))
    %>% mutate(pred = mu + b_i + b_u)
    %>% pull(pred)

  return(RMSE(predicted_ratings, testset$rating))
}
```

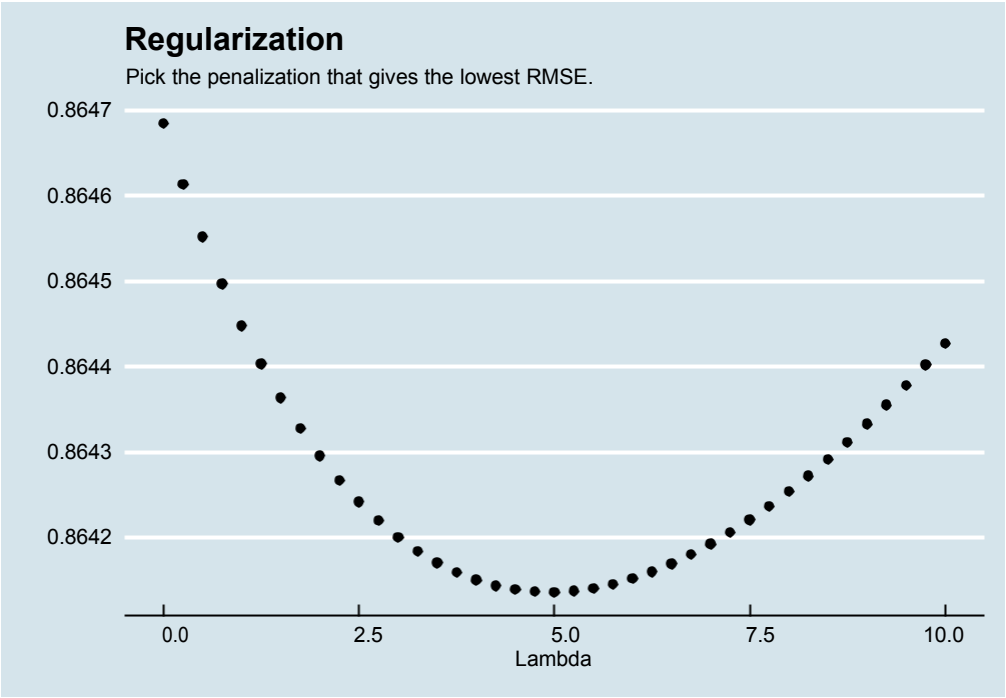


Figure 3.3: Choose the lambda that minimizes RMSE.

```
# Define a set of lambdas to
tune lambdas <- seq(0, 10, 0.25)

# Update RMSES table
rmses <- sapply(lambdas,
                regularization,
                trainset = train_set,
                testset = test_set)

# Plot the lambda x RMSE
tibble(Lambda = lambdas, RMSE = rmses) %>
  % ggplot(aes(x = Lambda, y = RMSE)) +
    geom_point() +
    ggtitle("Regularization",
            subtitle = "Pick the penalization that gives the lowest
    RMSE.") + theme_economist()
```

Next, we apply the best λ to the linear model.

```
# We pick the lambda that returns the lowest
RMSE. lambda <- lambdas[which.min(rmses)]

# Then, we calculate the predicted rating using the best parameters
```

```r
# achieved from regularization.
mu <- mean(train_set$rating)

# Movie effect (bi)
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

# User effect (bu)
b_u <- train_set %>%
  left_join(b_i, by="movieId")
  %>% group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

# Prediction
y_hat_reg <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Update the result table
result <- bind_rows(result,
                    tibble(Method = "Regularized bi and bu",
                           RMSE = RMSE(test_set$rating, y_hat_reg),
                           MSE = MSE(test_set$rating, y_hat_reg),
                           MAE = MAE(test_set$rating, y_hat_reg)))

# Regularization made a small improvement
in RMSE. result
```

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5012445	2.2537350	1.1675974
Mean	1.0600537	1.1237139	0.8551636
Mean + bi	0.9429615	0.8891764	0.7370384
Mean + bi + bu	0.8646843	0.7476789	0.6684369
Regularized bi and bu	0.8641362	0.7467313	0.6687070

3.5 Matrix Factorization

Matrix factorization approximates a large usermovie matrix into the product of two smaller dimension ma trices. Information in the train set is stored in tidy format, with one observation per row, so it needs to be converted to the usermovie matrix before using matrix factorization. This code executes this transformation.

```
train_data <- train_set %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()
```

The code above uses more memory than a commodity laptop is able to process, so we use an alternative method: the `recosystem` package, which provides the complete solution for a recommendation system using matrix factorization.

The package vignette[1] describes how to use `recosystem`:

Usage of `recosystem`

The usage of **`recosystem`** is quite simple, mainly consisting of the following steps:

1. Create a model object (a Reference Class object in R) by calling `Reco()`.
2. (Optionally) call the `$tune()` method to select best tuning parameters along a set of candidate values.
3. Train the model by calling the `$train()` method. A number of parameters can be set inside the function, possibly coming from the result of `$tune()`.
4. (Optionally) export the model via `$output()`, i.e. write the factorization matrices P and Q into files or return them as R objects.
5. Use the `$predict()` method to compute predicted values.

```
if(!require(recosystem))
  install.packages("recosystem", repos = "http://cran.us.r-project.org")
set.seed(123, sample.kind = "Rounding") # This is a randomized algorithm

# Convert the train and test sets into recosystem input format
train_data <- with(train_set, data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating     = rating))
test_data <- with(test_set,    data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating     = rating))

# Create the model object
r <- recosystem::Reco()
```

1 https://cran.rproject.org/web/packages/recosystem/vignettes/introduction.html

```
# Select the best tuning parameters
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread = 4, niter = 10))

 # Train the algorithm
r$train(train_data, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter    tr_rmse       obj
##    0     0.9821 1.1039e+007
##    1     0.8750 8.9801e+006
##    2     0.8418 8.3363e+006
##    3     0.8201 7.9506e+006
##    4     0.8043 7.6849e+006
##    5     0.7924 7.4997e+006
##    6     0.7827 7.3630e+006
##    7     0.7743 7.2462e+006
##    8     0.7670 7.1534e+006
##    9     0.7605 7.0755e+006
##   10     0.7549 7.0073e+006
##   11     0.7497 6.9523e+006
##   12     0.7453 6.9007e+006
##   13     0.7411 6.8563e+006
##   14     0.7372 6.8145e+006
##   15     0.7337 6.7818e+006
##   16     0.7305 6.7493e+006
##   17     0.7275 6.7204e+006
##   18     0.7248 6.6967e+006
##   19     0.7221 6.6718e+006
```

```
# Calculate the predicted values
y_hat_reco <-  r$predict(test_data, out_memory())
head(y_hat_reco, 10)
```

```
#   [1] 4.942627 3.792825 3.318276 3.157988 3.613130 3.990181 3.534084
#   [8] 3.351265 3.980333 3.157587
```

Matrix factorization improved substantially the RMSE.

```
result <- bind_rows(result,
                    tibble(Method = "Matrix Factorization - recosystem",
                           RMSE = RMSE(test_set$rating, y_hat_reco), MSE
                           = MSE(test_set$rating, y_hat_reco),
                           MAE  = MAE(test_set$rating, y_hat_reco)))
result
```

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5012445	2.2537350	1.1675974
Mean	1.0600537	1.1237139	0.8551636
Mean + bi	0.9429615	0.8891764	0.7370384
Mean + bi + bu	0.8646843	0.7476789	0.6684369
Regularized bi and bu	0.8641362	0.7467313	0.6687070
Matrix Factorization recosystem	0.7857474	0.6173989	0.6052006

3.6 Final Validation

As we can see from the result table, regularization and matrix factorization achieved the target RMSE. So, finally we train the complete `edx` set with both models and calculate the RMSE in the `validation` set. The project goal is achieved if the RMSE stays below the target.

3.6.1 Linear Model With Regularization

During the training and testing phase, the linear model with regularization achieved the target RMSE with a small margin. Here we do the final validation with the `validation` set.

```
mu_edx <- mean(edx$rating)

# Movie effect (bi)
b_i_edx <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_edx)/(n()+lambda))

# User effect (bu)
b_u_edx <- edx %>%
  left_join(b_i_edx, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_edx)/(n()+lambda))
```

```
# Prediction
y_hat_edx <- validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  mutate(pred = mu_edx + b_i + b_u) %>%
  pull(pred)

# Update the results table
result <- bind_rows(result,
                    tibble(Method = "Final Regularization (edx vs validation)",
                           RMSE = RMSE(validation$rating, y_hat_edx),
                           MSE = MSE(validation$rating, y_hat_edx),
                           MAE = MAE(validation$rating, y_hat_edx)))

# Show the RMSE
improvement result
```

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5012445	2.2537350	1.1675974
Mean	1.0600537	1.1237139	0.8551636
Mean + bi	0.9429615	0.8891764	0.7370384
Mean + bi + bu	0.8646843	0.7476789	0.6684369
Regularized bi and bu	0.8641362	0.7467313	0.6687070
Matrix Factorization recosystem	0.7857474	0.6173989	0.6052006
Final Regularization (edx vs validation)	0.8648177	0.7479097	0.6693494

As expected, the RMSE calculated on the `validation` set (0.8648177) is lower than the target of 0.8649 and slightly higher than the RMSE of the test set (0.8641362).

Top 10 best movies

```
validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  mutate(pred = mu_edx + b_i + b_u) %>%
  arrange(-pred) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

title
Usual Suspects, The (1995)
Shawshank Redemption, The (1994)
Shawshank Redemption, The (1994)
Shawshank Redemption, The (1994)
Eternal Sunshine of the Spotless Mind (2004)
Star Wars: Episode IV A New Hope (a.k.a. Star Wars) (1977)
Schindler's List (1993)
Donnie Darko (2001)
Star Wars: Episode VI Return of the Jedi (1983)
Schindler's List (1993)

Top 10 worst movies

```
validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  mutate(pred = mu_edx + b_i + b_u) %>%
  arrange(pred) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

title
Battlefield Earth (2000)
Police Academy 4: Citizens on Patrol (1987)
Karate Kid Part III, The (1989)
PokÃ©mon Heroes (2003)
Turbo: A Power Rangers Movie (1997)
Kazaam (1996)
PokÃ©mon Heroes (2003)
Free Willy 3: The Rescue (1997)
Shanghai Surprise (1986)
Steel (1997)

3.6.2 Matrix Factorization

The initial test shows that matrix factorization gives the best RMSE. Now it's time to validate with the entire `edx` and `validation` sets.

```
set.seed(1234, sample.kind = "Rounding")

# Convert 'edx' and 'validation' sets to recosystem input
format edx_reco <- with(edx, data_memory(user_index = userId,
                                         item_index = movieId,
                                         rating = rating))
validation_reco   <-with(validation, data_memory(user_index = userId,
                                                 item_index = movieId,
                                                 rating = rating))

# Create the model object
r <- recosystem::Reco()

# Tune the parameters
opts <- r$tune(edx_reco, opts = list(dim = c(10, 20, 30),
                                     lrate = c(0.1, 0.2),
                                     costp_l2 = c(0.01, 0.1),
                                     costq_l2 = c(0.01, 0.1),
                                     nthread = 4, niter = 10))

# Train the model
r$train(edx_reco, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter    tr_rmse       obj
##    0     0.9724 1.2014e+007
##    1     0.8718 9.8722e+006
##    2     0.8379 9.1549e+006
##    3     0.8168 8.7449e+006
##    4     0.8017 8.4735e+006
##    5     0.7902 8.2830e+006
##    6     0.7805 8.1279e+006
##    7     0.7724 8.0075e+006
##    8     0.7652 7.9098e+006
##    9     0.7591 7.8279e+006
##   10     0.7540 7.7606e+006
##   11     0.7494 7.7028e+006
##   12     0.7452 7.6511e+006
##   13     0.7415 7.6084e+006
##   14     0.7381 7.5692e+006
##   15     0.7350 7.5366e+006
##   16     0.7322 7.5062e+006
##   17     0.7296 7.4776e+006
##   18     0.7272 7.4536e+006
##   19     0.7249 7.4300e+006
```



```
# Calculate the prediction
```

```
y_hat_final_reco <- r$predict(validation_reco, out_memory())

# Update the result table
final_result <-
bind_rows(result,
                    tibble(Method = "Final Matrix Factorization - recosystem",
                           RMSE = RMSE(validation$rating, y_hat_final_reco),
                           MSE = MSE(validation$rating, y_hat_final_reco), MAE
                           = MAE(validation$rating, y_hat_final_reco)))
```

The final RMSE with matrix factorization is 0.7831304, 9.4% better than the linear model with regularization (0.8648177).

```
# Show the RMSE
improvement
final_result
```

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5012445	2.2537350	1.1675974
Mean	1.0600537	1.1237139	0.8551636
Mean + bi	0.9429615	0.8891764	0.7370384
Mean + bi + bu	0.8646843	0.7476789	0.6684369
Regularized bi and bu	0.8641362	0.7467313	0.6687070
Matrix Factorization recosystem	0.7857474	0.6173989	0.6052006
Final Regularization (edx vs validation)	0.8648177	0.7479097	0.6693494
Final Matrix Factorization recosystem	0.7831304	0.6132931	0.6032573

Chapter 4

Conclusion

We got the final predictions using various series of analysis across our dataset."final_result" gives us the predictions we want.

Limitation

This process of data analysis takes a certain amount of time to load, especially during traing the model. Make sure that your R's memory limit is set to 10000 for the ease of execution of the R code.