Movie Rating Prediction Project; Movielens Dataset

Adrianos Evangelos Botsios

9 July 2023

Introduction

The focus of this paper is to analyze data, develop and train a movie recommendation machine learning algorithm that will predict movie ratings. The data to do this will be provided by the Movielens dataset. To evaluate the accuracy of the algorithm the Root Mean Squared Estimate (RMSE) will be used. The goal is to obtain a RMSE of 0.86490 or lower. We will be creating and comparing 8 models and construct 2 final models.

This paper will present the analysis of data, the building of machine learning models, comparing rmse's, and conclusions.

The dataset:

A preliminary review and analysis of the dataset will be conducted prior to building the model in order to better understand the data. The data can be downloaded with the following code.

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(tidyverse)
library(caret)
library(data.table)
```

```
##
## Attaching package: 'data.table'
##
## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))
```

```
movielens <- left_join(ratings, movies, by = "movieId")


# Final hold-out test 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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

```
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

We can see the edx dataset in tidy format which has 6 columns of "userId", "movieId", "rating", "timestamp", "title" and "genres".

```
tibble(head(edx, 5))
```

```
## # A tibble: 5 x 6
##   userId movieId rating timestamp title                         genres
##    <int>   <int>  <dbl>     <int> <chr>                         <chr>
## 1      1     122      5 838985046 Boomerang (1992)              Comedy|Romance
## 2      1     185      5 838983525 Net, The (1995)               Action|Crime|Th~
## 3      1     292      5 838983421 Outbreak (1995)               Action|Drama|Sc~
## 4      1     316      5 838983392 Stargate (1994)               Action|Adventur~
## 5      1     329      5 838983392 Star Trek: Generations (1994) Action|Adventur~
```

It is important to notice the dimensions of the dataset to get an understanding of its size.

```
dim(edx)
```

```
## [1] 9000055       6
```

Then we can explore the dataset through the summary and see that the ratings of movies range from 5 to 0.5.

```
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
##
##
##
```

The number of distinct movies, users, and ratings provides more information on how the data set should be approached, and if it is sufficient for a recommendation algorithm.

```
edx %>%
  summarize(number_of_users = n_distinct(userId),
            number_of_movies = n_distinct(movieId),
            number_of_ratings = n_distinct(genres))
```

```
##   number_of_users number_of_movies number_of_ratings
## 1           69878            10677               797
```

It is important to look at popularity among different movie genres and the ratings they receive. Drama is the genre with the highest rating.

```
genres <- c("Comedy", "Drama", "Romance", "Thriller")
sapply(genres, function(g) {
  sum(str_detect(edx$genres, g))
})
```

```
##   Comedy    Drama  Romance Thriller
##  3540930  3910127  1712100  2325899
```

Then we can see the top 10 movies with the largest ratings.

```
edx %>%
  group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  top_n(10)
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
## Selecting by count
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                     count
##      <int> <chr>                                                     <int>
```

```
##  1     296 Pulp Fiction (1994)                                          31362
##  2     356 Forrest Gump (1994)                                          31079
##  3     593 Silence of the Lambs, The (1991)                             30382
##  4     480 Jurassic Park (1993)                                         29360
##  5     318 Shawshank Redemption, The (1994)                             28015
##  6     110 Braveheart (1995)                                            26212
##  7     457 Fugitive, The (1993)                                         25998
##  8     589 Terminator 2: Judgment Day (1991)                            25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10     150 Apollo 13 (1995)                                             24284
## # i 10,667 more rows
```

We have to create our train and test sets, with the train set being 80% of the data while the test set being 20%.

```
set.seed(1, sample.kind="Rounding")
test_set_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_set_index,]
test_set <- edx[test_set_index,]
```

Also, it is important to make sure there aren't any users in the test set that do not appear in the train set or vice-versa.

```
test_set <- test_set %>%
  semi_join(train_set, by = "userId") %>%
  semi_join(train_set, by = "movieId")
```

We will build 8 models and compare them to see which one is the highest performing.

First we will need to define the Root Mean Squared Estimate (RMSE) to keep track of our error score. The RMSE function will measure the accuracy of our models and the error, which should be less than 0.86490. The RMSE function can be defined like this:

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

In the above, n is the number of user and movie combinations, y_{u,i} is the rating for movie i, by user u, and $\hat{y}$_{u,i} is the prediction for the rating. We will define this function in R.

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

Creation of Models and Analysis

Model 1

We will start with the first model which will be a simple way to predict the ratings. The term mu will be introduced, which is the average value of all ratings. Also the term epsilon_{u,i} will be included and it is the independent errors that are sampled from the same distribution centered at 0. The model looks like this:

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

This model will be written in code, used to predict the rmse which will be around 1.059904, and saved in a table for the rmse scores. When the rmse is above 1, typically the prediction is off by 1 star which is not very good.

```
# Define mu
mu <- mean(train_set$rating)
# obtain the RMSE
mu_rmse <- RMSE(test_set$rating, mu)
mu_rmse
```

```
## [1] 1.059904
```

```
# save it into a table
rmse_model_results <- tibble(method = "The Average of Ratings", rmse = mu_rmse)
```

Model 2

The 2nd model takes into consideration the genres, because as it was seen earlier they have a large difference in ratings between them. The model finds the average rating for the different genres and uses that to predict the ratings though the variable g_i. The model will look like this:

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

With this method the rmse is 1.017801, which is a slight improvement compared to the previous model.

```
# Define the variable g_i
genre_averages <- train_set %>%
  group_by(genres) %>%
  summarize(g_i = mean(rating))
# Use the variable g_i to predict the ratings.
genre_average_pred <- test_set %>%
  mutate(Match = match(genres, genre_averages$genres), prediction = genre_averages$g_i[Match]) %>%
  pull(prediction)
# rmse of genre model
genre_average_rmse <- RMSE(test_set$rating, genre_average_pred)
genre_average_rmse
```

```
## [1] 1.017801
```

```
# Save rmse into
rmse_model_results <- bind_rows(rmse_model_results, tibble(method = "The Genre Rating Effect",
                                                           rmse = genre_average_rmse))
```

Model 3

The 3rd model will use the movie effect. Some movies are rated higher than others, so a new term, b_i should be added to the model. It is going to represent the average rating for a movie i, that the rating is going to be predicted for. Hence the model looks like this:

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

b_i is defined as the average of the difference between the movie rating Y_{u,i} and mu. After, the process will be the same as the other models where the model is used to predict the ratings, get an rmse and add it to the table. With the new term the rmse will improve and decrease to 0.9437429.

```
# Define the variable b_i
movie_averages <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
# Predict the ratings using b_i
movie_averages_pred <- mu + test_set %>%
  left_join(movie_averages, by = 'movieId') %>%
  pull(b_i)
# Calculate the rmse
movie_average_rmse <- RMSE(test_set$rating, movie_averages_pred)
movie_average_rmse
```

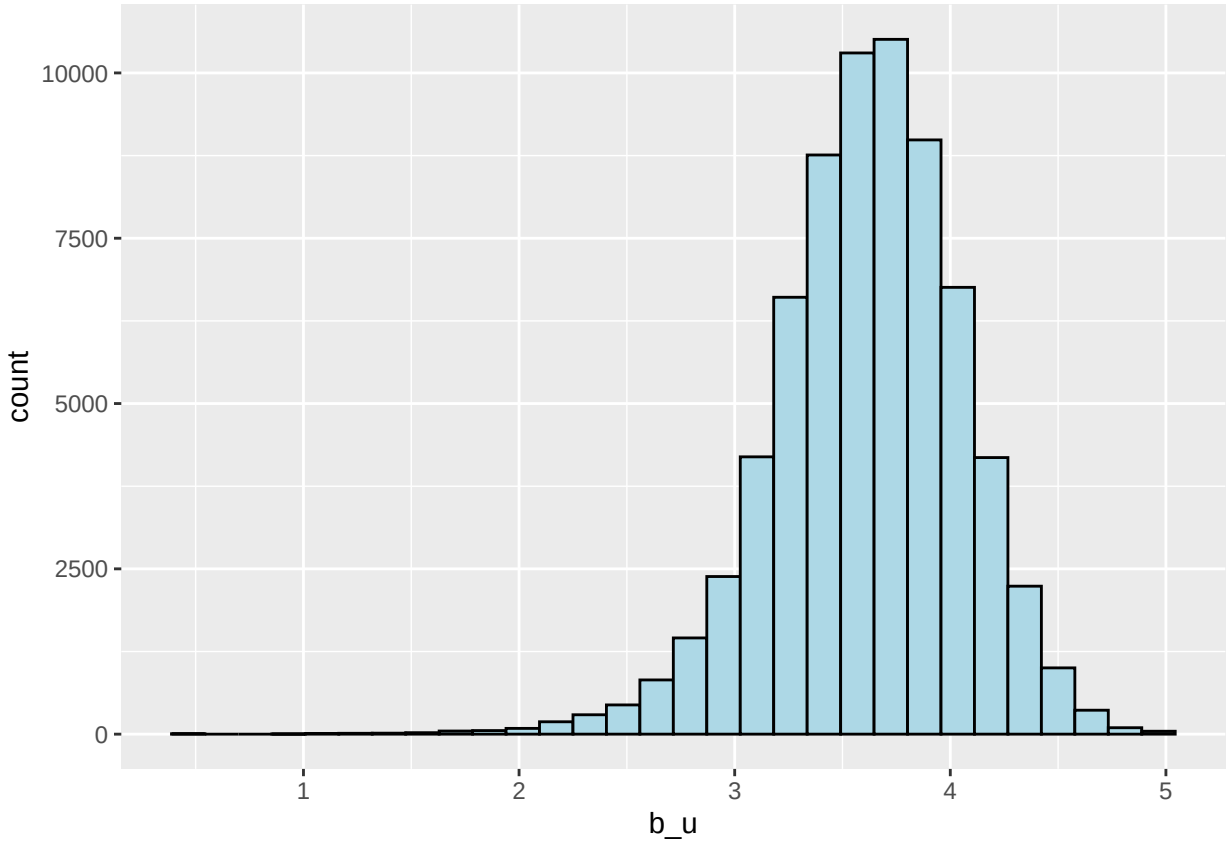
```
## [1] 0.9437429
```

```
# Save to the table of rmse's
rmse_model_results <- bind_rows(rmse_model_results, tibble(method = "The Movie Rating Effect",
                                                           rmse = movie_average_rmse))
```

Model 4:

Next we will have to use a different method to introduce a term to our model. First we can see if there are users which have different ways to rate movies. To explore the data we will find the average rating for a user u, given that they have rated over 100 movies and create a histogram on it.

```
# Hisogram for the average rating of users that have rated over 100 movies
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black", fill = "light blue")
```



From this graph we can see that there is substantial variability across users. That is due to the fact that some users rate movies with a low score because they are cranky, others rate every movie high, while other users are somewhere in the middle. This implies that we should have a term b_u, that will be the user effect. As a result the model will look like this.

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

First we will define the new b_u term by taking the average of the movie rating Y_{u,i} subtracted by mu and b_i. Then combine all the terms into one model that will be used to predict the ratings. This will result in a large improvement of the rmse which drops to 0.865932.

```
# Define the b_u term
user_averages <- train_set %>%
  left_join(movie_averages, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
# Predict ratings with the new model
user_average_pred <- test_set %>%
  left_join(movie_averages, by='movieId') %>%
  left_join(user_averages, by='userId') %>%
  mutate(prediction = mu + b_i + b_u) %>%
  pull(prediction)
# Compute the rmse
user_averages_rmse <- RMSE(test_set$rating, user_average_pred)
user_averages_rmse
```

```
## [1] 0.865932
```

```
# Save that into the rmse table
rmse_model_results <- bind_rows(rmse_model_results,
                                tibble(method = "The Movie and User Rating Effect",
                                       rmse = user_averages_rmse))
```

Model 5

We will still need to improve the rmse as it is not lower than our original goal. So we will look at the 10 largest mistakes that caused the largest rmse.

```
# 10 largest mistakes made
test_set %>%
  left_join(movie_averages, by = "movieId") %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  select(title, residual) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	residual
From Justin to Kelly (2003)	4.154762
Time Changer (2002)	4.000000
Shawshank Redemption, The (1994)	-3.957318
Shawshank Redemption, The (1994)	-3.957318
Shawshank Redemption, The (1994)	-3.957318
Shawshank Redemption, The (1994)	-3.957318
Shawshank Redemption, The (1994)	-3.957318
Shawshank Redemption, The (1994)	-3.957318
Shawshank Redemption, The (1994)	-3.957318
Children Underground (2000)	-3.928571

From the list it can be inferred that the movies are uncertain and in the model they obtained large predictions. To analyze further we will look at the top 10 worst and best movies caused by the estimates of the b_i, movie effect.

To see the movie titles we will create a database with the titles and movieId.

```
# Database with the movie id's and movie titles
movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()
```

We can see the 10 best movies.

```
#
movie_averages %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Hellhounds on My Trail (1999)	1.487518
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.487518
Satan's Tango (Sátántangó) (1994)	1.487518
Shadows of Forgotten Ancestors (1964)	1.487518
Money (Argent, L') (1983)	1.487518
Fighting Elegy (Kenka erejii) (1966)	1.487518
Sun Alley (Sonnenallee) (1999)	1.487518
Aerial, The (La Antena) (2007)	1.487518
Blue Light, The (Das Blaue Licht) (1932)	1.487518
More (1998)	1.404184

We can see the 10 worst movies.

```
# Top 10 worst movies
movie_averages %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(b_i) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Besotted (2001)	-3.012482
Confessions of a Superhero (2007)	-3.012482
War of the Worlds 2: The Next Wave (2008)	-3.012482
SuperBabies: Baby Geniuses 2 (2004)	-2.749982
From Justin to Kelly (2003)	-2.667244
Legion of the Dead (2000)	-2.637482
Disaster Movie (2008)	-2.637482
Hip Hop Witch, Da (2000)	-2.603391
Criminals (1996)	-2.512482
Mountain Eagle, The (1926)	-2.512482

Again they are obscure movies, so we will look at the number of ratings that the best movies received.

```
# The number of ratings each movie received
train_set %>%
  dplyr::count(movieId) %>%
  left_join(movie_averages) %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining with `by = join_by(movieId)`
```

title	b_i	n
Hellhounds on My Trail (1999)	1.487518	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.487518	3
Satan's Tango (Sátántangó) (1994)	1.487518	2
Shadows of Forgotten Ancestors (1964)	1.487518	1
Money (Argent, L') (1983)	1.487518	1
Fighting Elegy (Kenka erejii) (1966)	1.487518	1
Sun Alley (Sonnenallee) (1999)	1.487518	1
Aerial, The (La Antena) (2007)	1.487518	1
Blue Light, The (Das Blaue Licht) (1932)	1.487518	1
More (1998)	1.404184	6

From this we can see that they all were rated very few times. A small number of users rating the movies results in more uncertainty and larger estimates of b_i, positive or negative.

To solve this problem we will use regularization. Regularization will constrain the total variability of the effect size by penalizing large estimates coming from small sample sizes. We will add a penalty for large values of b to the sum of squares equations that we will minimize. However, having numerous large b's makes it harder to minimize the equation. Consequently, to estimate the b's we will not minimize the residual sum of square but the following equation where the penalty is lambda:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i\right)^2+\lambda\sum_i b_i^2$$

The Residual Sum of Squares is:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i\right)^2$$

The penalty term that increases when there are numerous b's which are large is:

$$\lambda\sum_i b_i^2$$

Afterwards through calculus it can be concluded that the values of b that minimize the equation above are given by this formula, where n_i is the number of ratings b, for movie i:

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

This formula can be used to regularize our model. When n_i is large and the estimate is stable, lambda will be ignored because "n_i + lambda" will be almost equal to n_i which will have a very small change. Yet, when the n_i is small the estimate of b_i will decrease towards zero and make b_i not have an effect on the model's prediction.

To find the optimal lambda we will create a sequence from 0 to 10 with an increment of 0.25. Then pass that sequence to a function that will compute the results of b_i and b_u through the formula shown above, and use different values of lambda. In the function will also be the model we have defined in order to compute the rmse. Then we will see the lambda that was the most effective in the regularization.

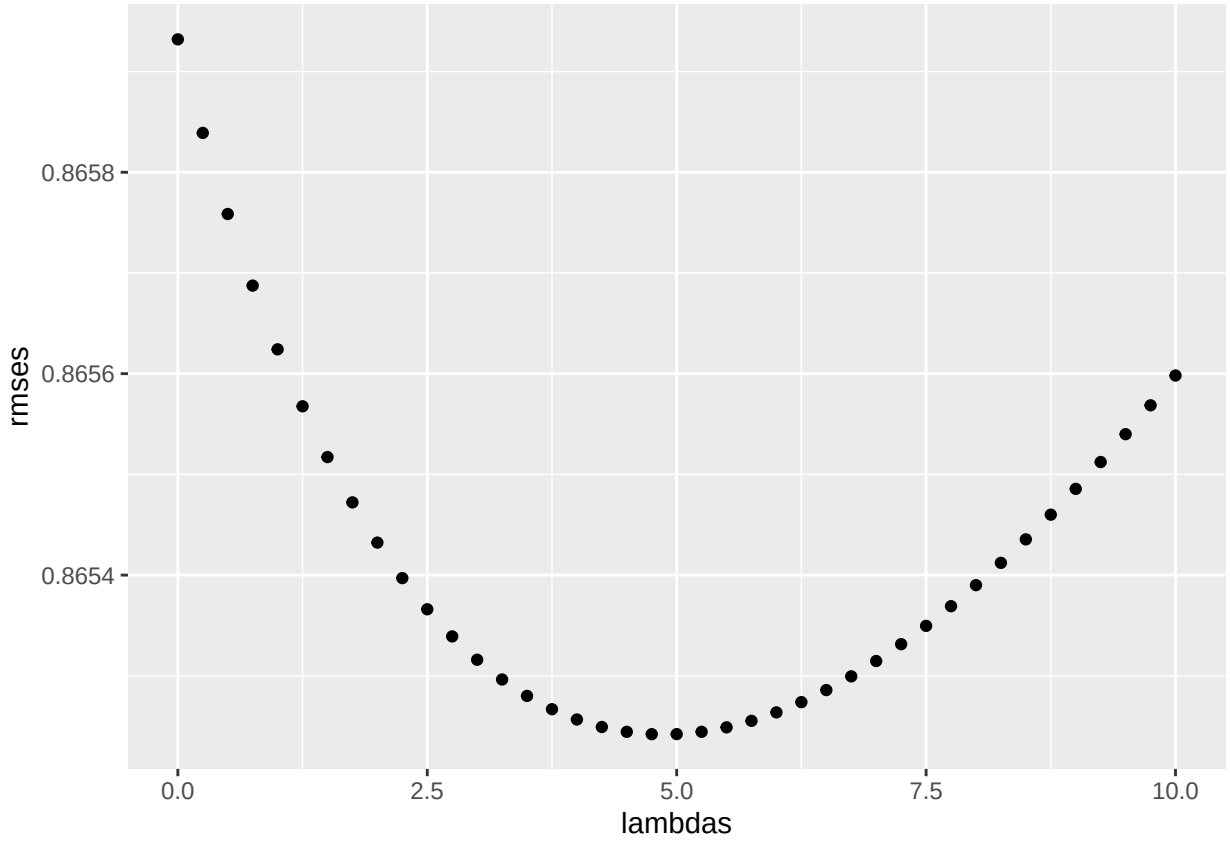
```
# We will find the lambda that best regularizes the users and movies
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
```

```
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(predicted_ratings, test_set$rating))
})
# The best regularizing lambda
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.75
```

We can also see a plot for the different values of lambdas and how they change the rmse.

```
qplot(lambdas, rmses)
```



Now we will re-compute the b_i but with the value of lambda which is 4.75 as suggested by our function and get an rmse of 0.943. Then we can add that to our table and compare the non-regularized b_i and the regularized b_i where we see a slight improvement from 0.9437429 to 0.9436987.

```
# Compute mu again
mu <- mean(train_set$rating)
# Compute b_i with lambda
movie_reg_averages <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
# Predict the ratings with the regularized b_i
movie_reg_average_pred <- test_set %>%
  left_join(movie_reg_averages, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
# Calculate the rmse
movie_reg_average_rmse <- RMSE(movie_reg_average_pred, test_set$rating)
movie_reg_average_rmse
```

```
## [1] 0.9436987
```

```
# Add the new rmse to the table
rmse_model_results <- bind_rows(rmse_model_results,
                                tibble(method = "The Regularized Movie Rating Effect",
                                       rmse = movie_reg_average_rmse))
rmse_model_results
```

```
## # A tibble: 5 x 2
##   method                                rmse
##   <chr>                                <dbl>
## 1 The Average of Ratings               1.06
## 2 The Genre Rating Effect              1.02
## 3 The Movie Rating Effect              0.944
## 4 The Movie and User Rating Effect     0.866
## 5 The Regularized Movie Rating Effect 0.944
```

Model 6:

Next we will also regularize the b_u and add it into the model. The model with the regularized b_i and b_u results in an rmse of 0.8652421 which is extremely close to the goal of 0.86490.

```
# Regularize the b_u
user_reg_averages <- train_set %>%
  left_join(movie_reg_averages, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda), n_i = n())
# We get the predictions
user_reg_averages_pred <- test_set %>%
  left_join(movie_reg_averages, by = 'movieId') %>%
  left_join(user_reg_averages, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
# We find the rmse
user_reg_averages_rmse <- RMSE(user_reg_averages_pred, test_set$rating)
user_reg_averages_rmse
```

```
## [1] 0.8652421
```

```
# We will now add the rmse to our table
rmse_model_results <- bind_rows(rmse_model_results,
                                tibble(method = "The Regularized Movie and User Rating Effect",
                                       rmse = user_reg_averages_rmse))
```

Model 7 & 8:

Now we will use a different approach called Matrix Factorization to try and decrease the rmse. Matrix factorization is a famous technique used in machine learning that helps factorize a matrix into different vectors that can be afterwards used as terms in a model.

To do this the recosystem library is going to be used, which provides useful functions to build and factorize models. Once the library is loaded, the training and test sets will be converted to the required format by the recosystem library.

```
# Load the library required for the matrix factorization
library(recosystem)
# Set the seed to 1
set.seed(1, sample.kind = "Rounding")
# Convert the training set into the required format by the recosystem library.
train_set_reco <-  with(train_set, data_memory(user_index = userId,
```

```
                                          item_index = movieId,
                                          rating     = rating,
                                          dates      = dates))
# Convert the training set into the required format by the recosystem library.
test_set_reco  <-  with(test_set,  data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating     = rating,
                                          dates      = dates))
```

Then we will create two models that are going to be stored in two different objects. This is due to the fact that we are going to be comparing 2 different sets of tuning parameters.

```
# Create the 2 models and store them in an object
recosystem_model_1 <-  recosystem::Reco()
recosystem_model_2 <-  recosystem::Reco()
```

We will define two different objects, with different tuning parameters which can be found online. The model with optimization_1 has the parameters from the article https://cran.r-project.org/web/packages/recosystem/recosystem.pdf by Yixuan Qiu. In the tuning parameters the first argument is dim which sets the number of latent factors the model will have. Then is the argument lrate which defines the learning rate of our model. The argument costp_l2 specifies the L2 regularization cost for the user factors, while the argument costq_l2 is the L2 regularization cost for the item factors. After is the argument nthread, which is given as an integer and is the number of threads for the parallel computing that are used. Lastly, the argument niter which is also given as an integer, is the number of iterations the model will have.

```
# This is the first set of parameters that we will try and save it to a variable called optimization_1
######### WARNING: The next line of code with optimization_1 took around 30 minutes to optimize!!
optimization_1 <- recosystem_model_1$tune(train_set_reco,
               opts = list(dim = c(10, 20, 30),          # dim is the number of latent factors.
                           lrate = c(0.1, 0.2),          # lrate is the learning rate.
                           costp_l2 = c(0.01, 0.1),      # costp_l2 is the L2 regularization cost for t
                           costq_l2 = c(0.01, 0.1),      # costq_l2 is the L2 regularization cost for t
                           nthread  = 4,                 # nthread is an integer and is the number of t
                           niter = 10))                  # niter is an integer and is the number of ite
```

The model with optimization_2 has the tuning parameters from the website https://rdrr.io/cran/recosystem/man/tune.html. The new optimization will include the argument dim, costp_l1 which is the L1 regularization cost for the user factors and costp_l2. Then it will have costq_l1 which is the L1 regularization cost for the user factor and then costq_l2. Lastly, it will have the argument lrate which specifies the learning rate of the model.

```
# This is the first set of parameters that we will try and save it to a variable called optimization_2
######### WARNING: The next line of code with optimization_2 took around 1:30 hours to optimize!!
optimization_2 <- recosystem_model_2$tune(train_set_reco, opts = list(dim = c(10L, 20L), # dim is the n
                                costp_l1 = c(0, 0.1),       # costp_l1 is the l1 regularization cost for
                                costp_l2 = c(0.01, 0.1),    # costp_l2 is the l2 regularization cost for
                                costq_l1 = c(0, 0.1),       # costq_l1 is the l1 regularization cost for
                                costq_l2 = c(0.01, 0.1),    # costq_l2 is the l2 regularization cost for
                                lrate    = c(0.01, 0.1))    # lrate is the learning rate.
)
```

Then we will use the tuning parameters that are selected by the optimization to train our 2 models.

```
# Train the algorithm with the optimized parameters parameters.
# optimization_1:
recosystem_model_1$train(train_set_reco,
                         opts = c(optimization_1$min))
```

```
## iter      tr_rmse          obj
##    0       0.9936   1.0045e+07
##    1       0.8793   8.0979e+06
##    2       0.8463   7.5131e+06
##    3       0.8237   7.1452e+06
##    4       0.8073   6.9025e+06
##    5       0.7949   6.7235e+06
##    6       0.7847   6.5896e+06
##    7       0.7759   6.4812e+06
##    8       0.7679   6.3886e+06
##    9       0.7611   6.3144e+06
##   10       0.7549   6.2470e+06
##   11       0.7493   6.1878e+06
##   12       0.7444   6.1412e+06
##   13       0.7398   6.0970e+06
##   14       0.7356   6.0591e+06
##   15       0.7318   6.0223e+06
##   16       0.7282   5.9923e+06
##   17       0.7250   5.9630e+06
##   18       0.7221   5.9381e+06
##   19       0.7193   5.9141e+06
```

```
# optimization_2:
recosystem_model_2$train(train_set_reco, opts = optimization_2$min)
```

```
## iter      tr_rmse          obj
##    0       0.9875   9.9596e+06
##    1       0.8800   8.0859e+06
##    2       0.8502   7.5369e+06
##    3       0.8311   7.2143e+06
##    4       0.8146   6.9749e+06
##    5       0.8014   6.7858e+06
##    6       0.7914   6.6496e+06
##    7       0.7834   6.5435e+06
##    8       0.7767   6.4586e+06
##    9       0.7710   6.3862e+06
##   10       0.7660   6.3232e+06
##   11       0.7618   6.2753e+06
##   12       0.7580   6.2315e+06
##   13       0.7546   6.1932e+06
##   14       0.7516   6.1604e+06
##   15       0.7489   6.1314e+06
##   16       0.7465   6.1036e+06
##   17       0.7443   6.0816e+06
##   18       0.7423   6.0598e+06
##   19       0.7404   6.0412e+06
```

Once the models have been trained they will be used to predict the ratings from the test_set_reco that was

defined above. The out_memory() function returns the result as an R object and is going to be used to compute the rmse.

```
# Find the predicted values using the test_data in reco format.
# out_memory() returns the result as an R object
# optimization_1 prediction
reco_model_1_pred <-  recosystem_model_1$predict(test_set_reco, out_memory())
#optimization_1 prediction
reco_model_2_pred <-  recosystem_model_2$predict(test_set_reco, out_memory())
```

Once we have obtained the predictions we will use them to find our model's rmse, and then add that to our rmse table. First we will find the rmse of the first model with tuning parameters from the variable optimization_1, which is around 0.7910392.

```
# We will then find their rmse's
# optimization_1
set.seed(1, sample.kind = "Rounding")
reco_model_1_rmse <- RMSE(test_set$rating, reco_model_1_pred)
reco_model_1_rmse
```

```
## [1] 0.7910392
```

```
# We get an rmse of 0.7902
# We will save that into our table for models
rmse_model_results <- bind_rows(rmse_model_results,
                                tibble(method = "Recosystem Model 1",
                                       rmse = reco_model_1_rmse))
```

Then we will find the rmse of the second model with the tuning parameters from the variable optimization_2, which is around 0.7948054.

```
# optimization_2
set.seed(1, sample.kind = "Rounding")
reco_model_2_rmse <- RMSE(test_set$rating, reco_model_2_pred)
reco_model_2_rmse
```

```
## [1] 0.7948054
```

```
# We get an rmse of 0.795
# We will save that into our table for models
rmse_model_results <- bind_rows(rmse_model_results,
                                tibble(method = "Recosystem Model 2",
                                       rmse = reco_model_2_rmse))
```

Now we can list our models and their rmse from the table to compare and see which one is the lowest and best performing.

```
# see the rmse table in increasing order
rmse_model_results %>% arrange(rmse)
```

```
## # A tibble: 8 x 2
##   method                                      rmse
##   <chr>                                      <dbl>
## 1 Recosystem Model 1                         0.791
## 2 Recosystem Model 2                         0.795
## 3 The Regularized Movie and User Rating Effect 0.865
## 4 The Movie and User Rating Effect           0.866
## 5 The Regularized Movie Rating Effect        0.944
## 6 The Movie Rating Effect                    0.944
## 7 The Genre Rating Effect                    1.02
## 8 The Average of Ratings                     1.06
```

It can be seen that the highest performing model with matrix factorization is the one with the parameters from optimization_1. It is important to note that the number of iterations in the model training have a large effect on the results. We will increase the number of iterations, but we have to be careful because if the iterations are too many the model might over fit. Over-fitting would result in a biased model that does well in only our test set but not good in the final holdout test set or in the real world. Now we will attempt to increase the iterations to around 35 which is a range when the rmse does not increase by a lot and also not by too little when using the test set, which proves that we are not over-fitting but also not under-fitting.

```
# re-train the recosystem_model_1 with 35 iterations
recosystem_model_1$train(train_set_reco,
                         opts = c(optimization_1$min, niter = 35))
```

```
## iter      tr_rmse          obj
##    0       0.9937   1.0047e+07
##    1       0.8798   8.0952e+06
##    2       0.8469   7.5133e+06
##    3       0.8249   7.1543e+06
##    4       0.8087   6.9099e+06
##    5       0.7960   6.7365e+06
##    6       0.7856   6.5942e+06
##    7       0.7763   6.4824e+06
##    8       0.7683   6.3929e+06
##    9       0.7612   6.3147e+06
##   10       0.7549   6.2466e+06
##   11       0.7492   6.1884e+06
##   12       0.7441   6.1412e+06
##   13       0.7394   6.0936e+06
##   14       0.7352   6.0553e+06
##   15       0.7313   6.0212e+06
##   16       0.7279   5.9895e+06
##   17       0.7245   5.9599e+06
##   18       0.7215   5.9347e+06
##   19       0.7187   5.9108e+06
##   20       0.7161   5.8890e+06
##   21       0.7138   5.8703e+06
##   22       0.7115   5.8522e+06
##   23       0.7094   5.8348e+06
##   24       0.7075   5.8211e+06
##   25       0.7056   5.8059e+06
##   26       0.7040   5.7922e+06
##   27       0.7023   5.7809e+06
```

```
##   28       0.7008   5.7690e+06
##   29       0.6994   5.7589e+06
##   30       0.6981   5.7493e+06
##   31       0.6968   5.7400e+06
##   32       0.6956   5.7311e+06
##   33       0.6945   5.7231e+06
##   34       0.6934   5.7150e+06
```

```
# Obtain the predictions
reco_model_1_pred <-  recosystem_model_1$predict(test_set_reco, out_memory())
set.seed(1, sample.kind = "Rounding")
```

In fact there is a small improvement and the rmse decreases to 0.7903591. We will stop at 35 iterations as that is a good number for our dataset size.

```
# re-calculate the rmse
reco_model_1_rmse <- RMSE(test_set$rating, reco_model_1_pred)
reco_model_1_rmse
```

```
## [1] 0.7903591
```

```
# Add that to our table
rmse_model_results <- bind_rows(rmse_model_results,
                                tibble(method = "Recosystem Model 1 with 35 iteraations",
                                       rmse = reco_model_1_rmse))
```

Final Models

To obtain our final rmse using the final_holdout_test set as it is our goal, we will create two final models. One model will be with the regularized user and movie effect and the other will be with the matrix factorization model with optimization_1 through the recosystem library.

Final Model 1

First we will run model 1 which is the average ratings and the regularized user and movie effect. The model equation is the following.

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

Now we will re-define the model and compute all the terms but use the edx data set, and then obtain the predictions and the rmse from the final_holdout_test set.

```
# Final Model 1
# This is the final model that will be tested by the the final holdout test
# average of the ratings
final_model_mu <- mean(edx$rating)
# the movie rating effect b_i
final_model_movie_reg_avg <- edx %>%
  group_by(movieId) %>%
```

```
  summarize(b_i = sum(rating - final_model_mu)/(n() +lambda), n_i =n())
# the user rating effect b_u
final_model_user_reg_avg <- edx %>%
  left_join(final_model_movie_reg_avg, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - final_model_mu - b_i)/(n()+lambda), n_i = n())
# the regularized movie and user effect
final_model_user_reg_avg_pred <- final_holdout_test %>%
  left_join(final_model_movie_reg_avg, by = 'movieId') %>%
  left_join(final_model_user_reg_avg, by = "userId") %>%
  mutate(pred = final_model_mu + b_i + b_u) %>%
  pull(pred)
set.seed(1, sample.kind = "Rounding")
```

```
# Get a final RMSE of 0.8648 on our first final model
FINAL_RMSE <- RMSE(final_holdout_test$rating, final_model_user_reg_avg_pred)
FINAL_RMSE
```

```
## [1] 0.8648201
```

The result of the rmse is 0.8648 which is lower than our goal of 0.86490 and the model is successful.

Final Model 2

Next we will run the model 2 which is the recosystem factorized matrix through using all the edx data set and testing it with the final_holdout_test set. We will convert the edx set and the final_holdout_test set to the format required by the recosystem. Then re-optimize the tuning parameters for the model and compute the final rmse with the final_holdout_test set.

```
# Final Model 2: Ricosystem Model
# Set the seed to 1
set.seed(1, sample.kind = "Rounding")
# Make the edx data to the format of the recosystem library
final_model_reco_edx_data  <-  with(edx,  data_memory(user_index = userId,
                                                      item_index = movieId,
                                                      rating     = rating,
                                                      dates      = dates))
set.seed(1, sample.kind = "Rounding")
# Make the final_holdout_test to the format of the recosystem library
final_model_reco_val_data  <-  with(final_holdout_test,  data_memory(user_index = userId,
                                                        item_index = movieId,
                                                        rating     = rating,
                                                        dates      = dates))
# Define our final recosystem model
final_model_recosystem <-  recosystem::Reco()
# Set the best preforming parameters to the final model
final_model_tuning_params <- final_model_recosystem$tune(final_model_reco_edx_data,
                             opts = list(dim = c(10, 20, 30),          # dim is the number of latent f
                                         lrate = c(0.1, 0.2),          # lrate is the learning rate.
                                         costp_l2 = c(0.01, 0.1),      # costp_l2 is the L2 regulariza
                                         costq_l2 = c(0.01, 0.1),      # costq_l2 is the L2 regulariza
```

```
                                          nthread  = 4,                  # nthread is an integer and is
                                          niter = 10))                  # niter is an integer and is th
set.seed(1, sample.kind = "Rounding")
# Train the final model with the parameters
final_model_recosystem$train(final_model_reco_edx_data,
                         opts = c(final_model_tuning_params$min, niter = 35))
```

```
## iter      tr_rmse          obj
##    0       0.9727   1.2025e+07
##    1       0.8726   9.8949e+06
##    2       0.8377   9.1635e+06
##    3       0.8156   8.7360e+06
##    4       0.8001   8.4596e+06
##    5       0.7884   8.2667e+06
##    6       0.7786   8.1110e+06
##    7       0.7709   7.9931e+06
##    8       0.7644   7.8977e+06
##    9       0.7588   7.8215e+06
##   10       0.7539   7.7583e+06
##   11       0.7496   7.7014e+06
##   12       0.7457   7.6540e+06
##   13       0.7421   7.6119e+06
##   14       0.7388   7.5727e+06
##   15       0.7357   7.5402e+06
##   16       0.7329   7.5108e+06
##   17       0.7302   7.4824e+06
##   18       0.7277   7.4574e+06
##   19       0.7254   7.4335e+06
##   20       0.7232   7.4140e+06
##   21       0.7212   7.3943e+06
##   22       0.7193   7.3751e+06
##   23       0.7175   7.3592e+06
##   24       0.7159   7.3432e+06
##   25       0.7143   7.3291e+06
##   26       0.7128   7.3149e+06
##   27       0.7114   7.3042e+06
##   28       0.7101   7.2909e+06
##   29       0.7088   7.2805e+06
##   30       0.7077   7.2700e+06
##   31       0.7065   7.2613e+06
##   32       0.7055   7.2533e+06
##   33       0.7045   7.2430e+06
##   34       0.7035   7.2356e+06
```

```
# Predict using the final recosystem model on the final_model_reco_val_data set
final_model_recosystem_pred <-  final_model_recosystem$predict(final_model_reco_val_data, out_memory())
set.seed(1, sample.kind = "Rounding")
```

```
# Get the final predictions of the RMSE
final_model_reco_rmse <- RMSE(final_holdout_test$rating, final_model_recosystem_pred)
final_model_reco_rmse
```

```
## [1] 0.7806114
```

The result of the rmse is 0.7806114 which is an incredibly good score compared to our goal.

Conclusion

To conclude, the most effective way to predict movie ratings is through the use of matrix factorization while the next best way is through model building. Two final models were constructed, one which used matrix factorization and the other that used the average rating and a regularized user and movie effect. The objective of this paper was to create a model that would get an rmse of 0.86490 or lower, which was achieved by both of the final models. Model 1 achieved an rmse of 0.8648 while Model 2 achieved an rmse of 0.7806114. For future papers, an improvement is to have access to a larger data set and more powerful computing because it took over 2 hours to run all the code and train the models. Additionally, it limited the ability to try many different and more complex models as they would take a lot longer to train on a personal computer. Data Science and Machine learning are very powerful tools that can help the world as long as people know how to use them correctly.