Movie Recommendation - Machine Learning Project HarvardX PH125.9x Data Science Capstone

Chi T. Dinh

2023-01-08

Introduction

The objective of this project is to develop a movie recommendation system using machine learning algorithm. The first step is to exam the data, visualize it, and then progressively develop a model that will satisfy the target accuracy of the residual mean squared error (RMSE) less than **0.8649.**

More details on Recommendation systems.

Background

In October 2006, Netflix offered a challenge to the data science community: improve our recommendation algorithm by 10% and win a million dollars.In September 2009, the winners were announced. A summary of how the winning algorithm was put together can be found here.

Some of the winning team's data analysis strategies will be explored in this project.

MovieLens Dataset

The Netflix data is not publicly available, but the GroupLens research lab generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users.

For this project the 10M MovieLensDataset Version from groupLens will be used. It contains 10 million ratings on 10,000 movies by 72,000 users.

The 10M MovieLens Dataset will be partitioned into two datasets: edx and validation (final_holdout_test). The edx dataset will be partitioned further into train and test datasets 90/10 ratio respectively.

The modeling approaches will be developed and evaluated using the edx partitions (train and test). The model with the best accuracy will be tested using the validation set (final_holdout_test).

Data Loading

This process may take a few minutes to run.

```
#####################################
# Data Loading
# Create edx dataset and final_holdout_test as a validate dataset
#####################################
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_holdout_test set contains 10% (p.01) of MovieLens data and serves as validation dataset.
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 final_holdout_test set are also in edx set
# Semi_join() function to return in one data frame that have matching values in another data fra
me.
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

validation <- final_holdout_test  # Assign final_holdout_test to validation object

# Add rows removed from final_holdout_test set back into edx set
# anti_join() function from the dplyr package to return all rows in one data frame that do not h
```

```
ave matching values in another data frame.
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

# rm() function to delete objects no longer required from the memory to reduce memory footprint
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Exploratory Data Analysis - MovieLens Dataset Overview

The edx dataset shows ~9 millions rows of observations and six(6) columns of variables. Each row of observation represents a rating given per user per movie.

- Six variables are userID, movieID, rating, timestamp, title, and genres.
- The userID, movieID, and timestamp (in seconds) are the whole numeric values of integer.
- The movieID and userId are the focal data points for the models.
- The string title contains the release year and it can be split from the title if it's useful for prediction.
- Genres are pipe-delimited string containing 18 unique genre categories including Action, Comedy, Sci-Fi, etc. The genre categories can be split if it affects rating outcome or useful for prediction.

Examine edx data structure:

```
str(edx)
```

```
## 'data.frame':    9000061 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 8389
83653 838984885 838983707 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (199
5)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thr
iller" ...
```

Edx first seven rows:

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

userId	movieId	rating	timestamp	title	genres
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

Number of unique users and movies:

```
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

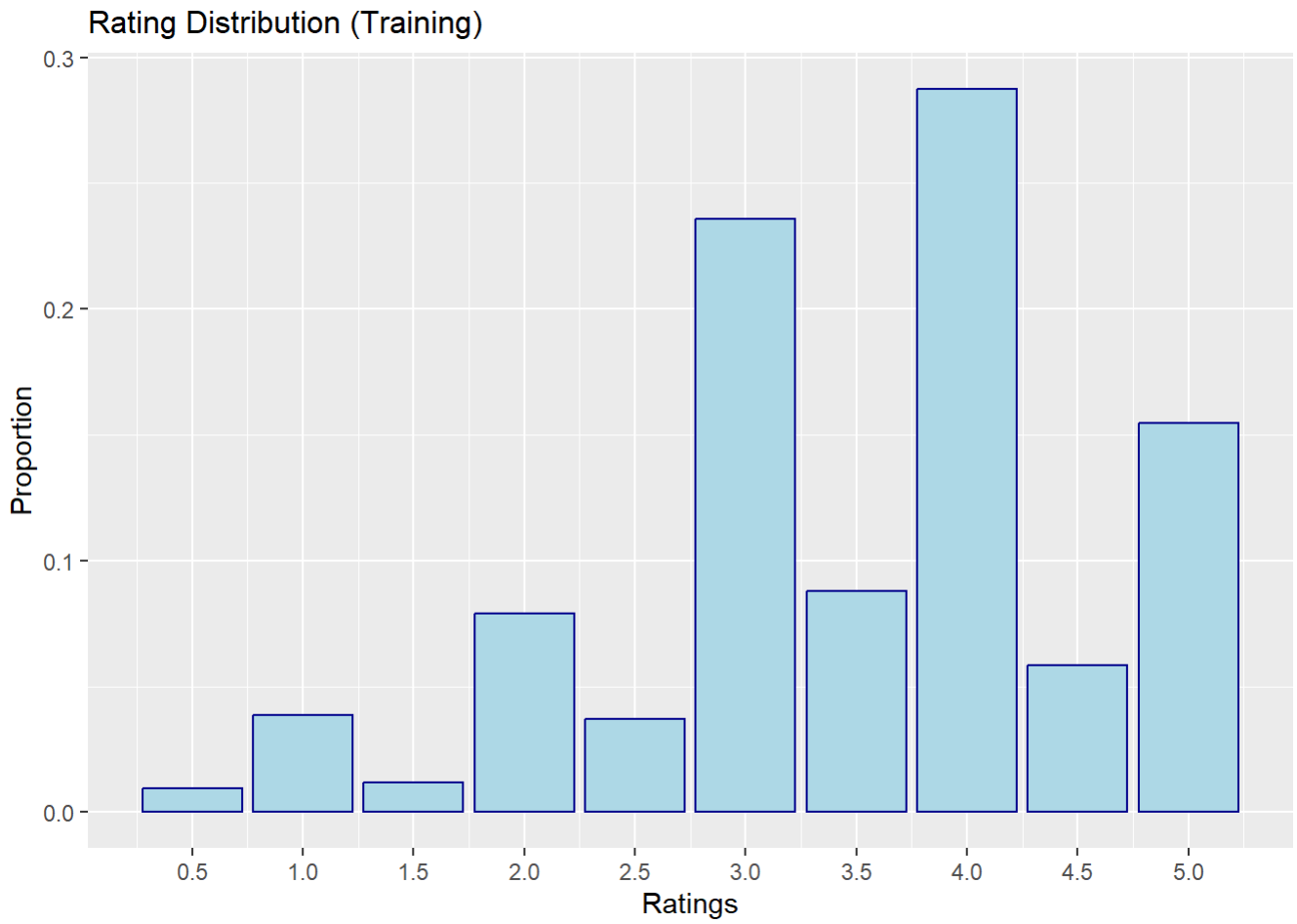
There are 10 different ratings lowest is 0.5 and highest 5.0. The average is 3.51, and there are no zero ratings.

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

```
## [1] 3.512464
```

Examine training distribution of the ratings further, 4.0 seems to be the most given rating.

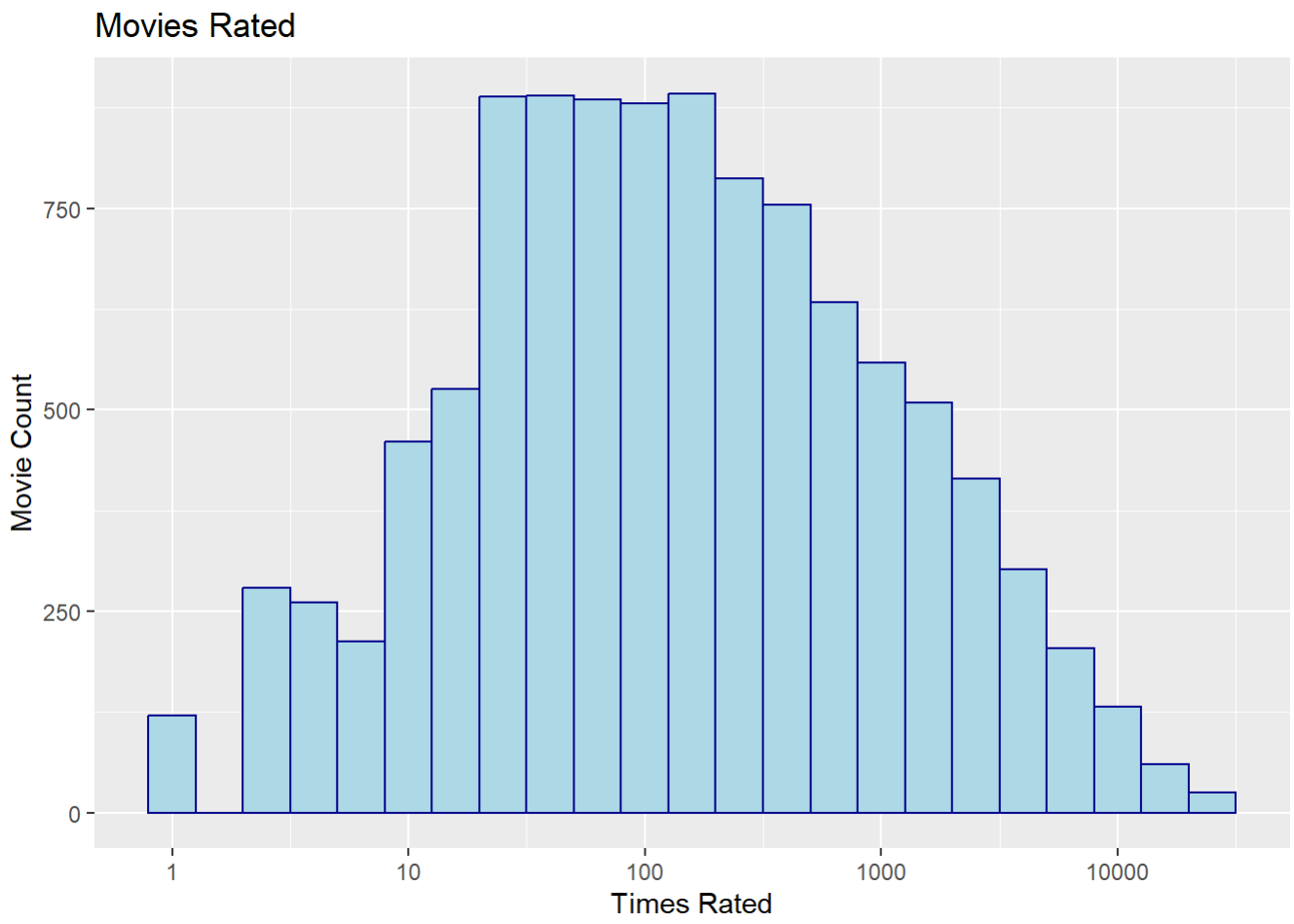
```
  edx %>%
    ggplot(aes(rating, y = ..prop..)) +
    geom_bar(color = "darkblue", fill="lightblue") +
    labs(x = "Ratings", y = "Proportion") +
    ggtitle("Rating Distribution (Training)") +
    theme(plot.title = element_text(size=12)) +
    scale_x_continuous(breaks = c(0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5))
```



A histogram is one of the popular graphing tools in data science. It helps to visualize whether the distribution is symmetric or skewed left or right. It can also show any outliners or gaps in the data.

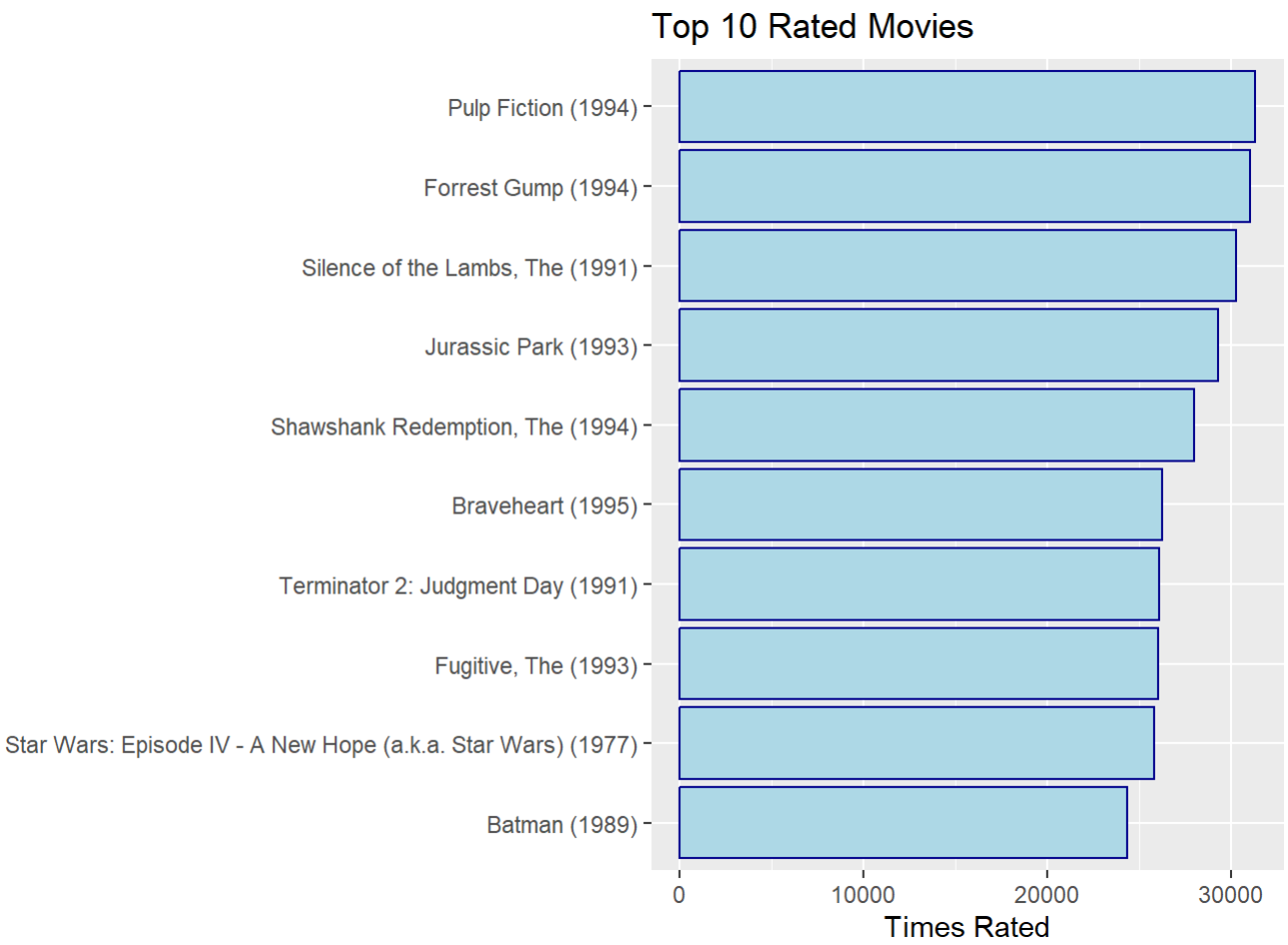
The Movies Rated histogram shows that some movies are rated more than others given that there are blockbuster movies watched by millions and artsy, independent movies watch by just a few per 33.7.1 Movielens data.

```
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, binwidth=0.2,  show.legend = FALSE, aes(fill = cut(n, 100)), fill="l
ightblue", color="darkblue") +
  scale_x_log10() +
  ggtitle("Movies Rated") +
  xlab("Times Rated") +
  ylab("Movie Count")
```



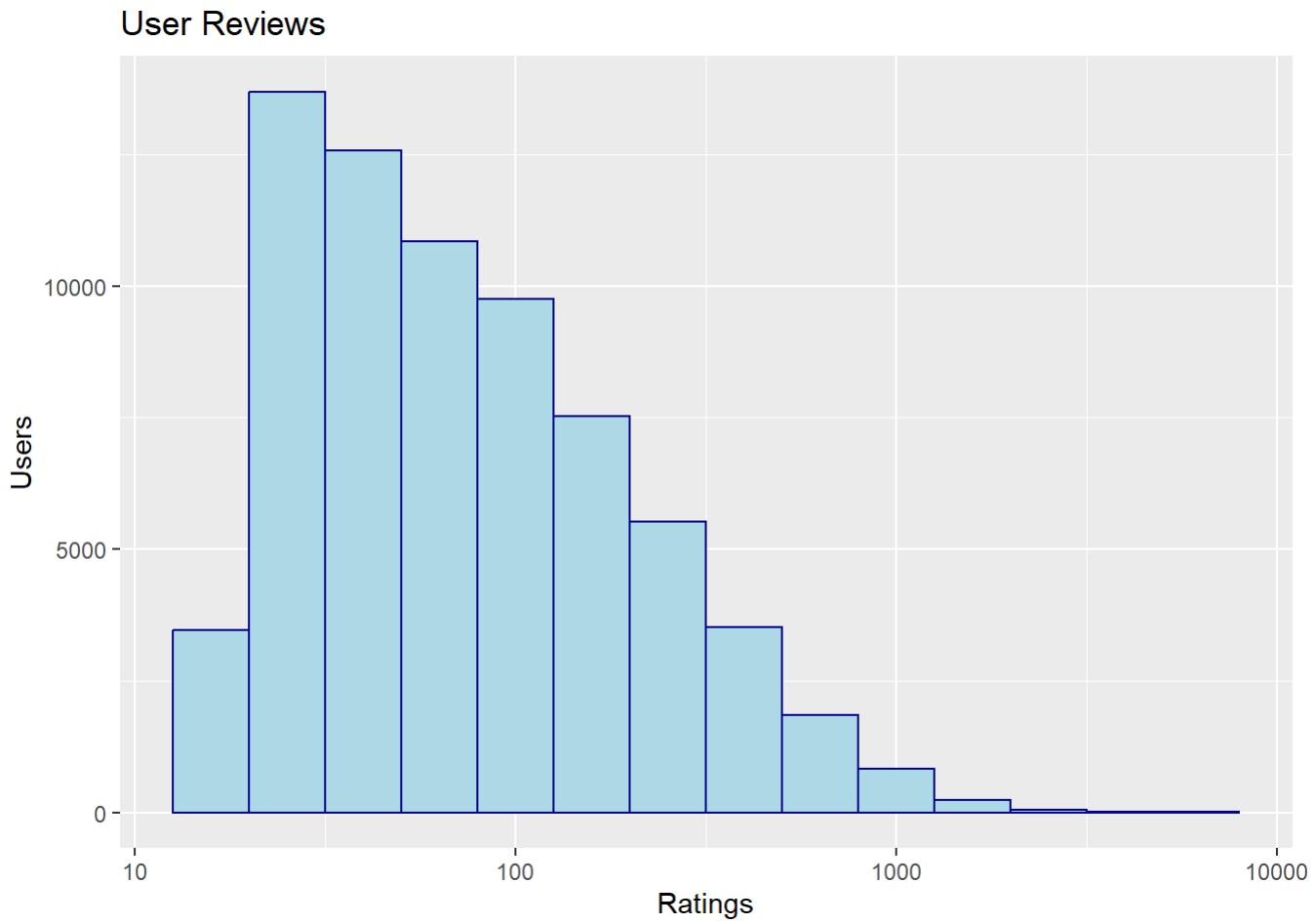
The following graph shows Top 10 most rated movies.

```
edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  arrange(-count) %>%
  top_n(10, count) %>%
  ggplot(aes(count, reorder(title, count))) +
  geom_bar(color = "darkblue", fill="lightblue", stat = "identity") +
  ggtitle("Top 10 Rated Movies") +
  xlab("Times Rated") +
  ylab("")
```



The majority of users rate between 10 and 100 movies, while some rate over 1,000. The graph shows a right skew in its distribution. This implies that some users are more active than other at rating movies.

```
  edx %>% count(userId) %>%
    ggplot(aes(n)) +
    geom_histogram(bins = 40, binwidth=0.2, show.legend = FALSE, aes(fill = cut(n, 100)), fill
="lightblue", color="darkblue") +
    scale_x_log10() +
    ggtitle("User Reviews") +
    xlab("Ratings") +
    ylab("Users")
```



Number of unique genres:

```
##   n_genres
## 1      797
```

Top 20 genres ordered by number of ratings in this dataset.

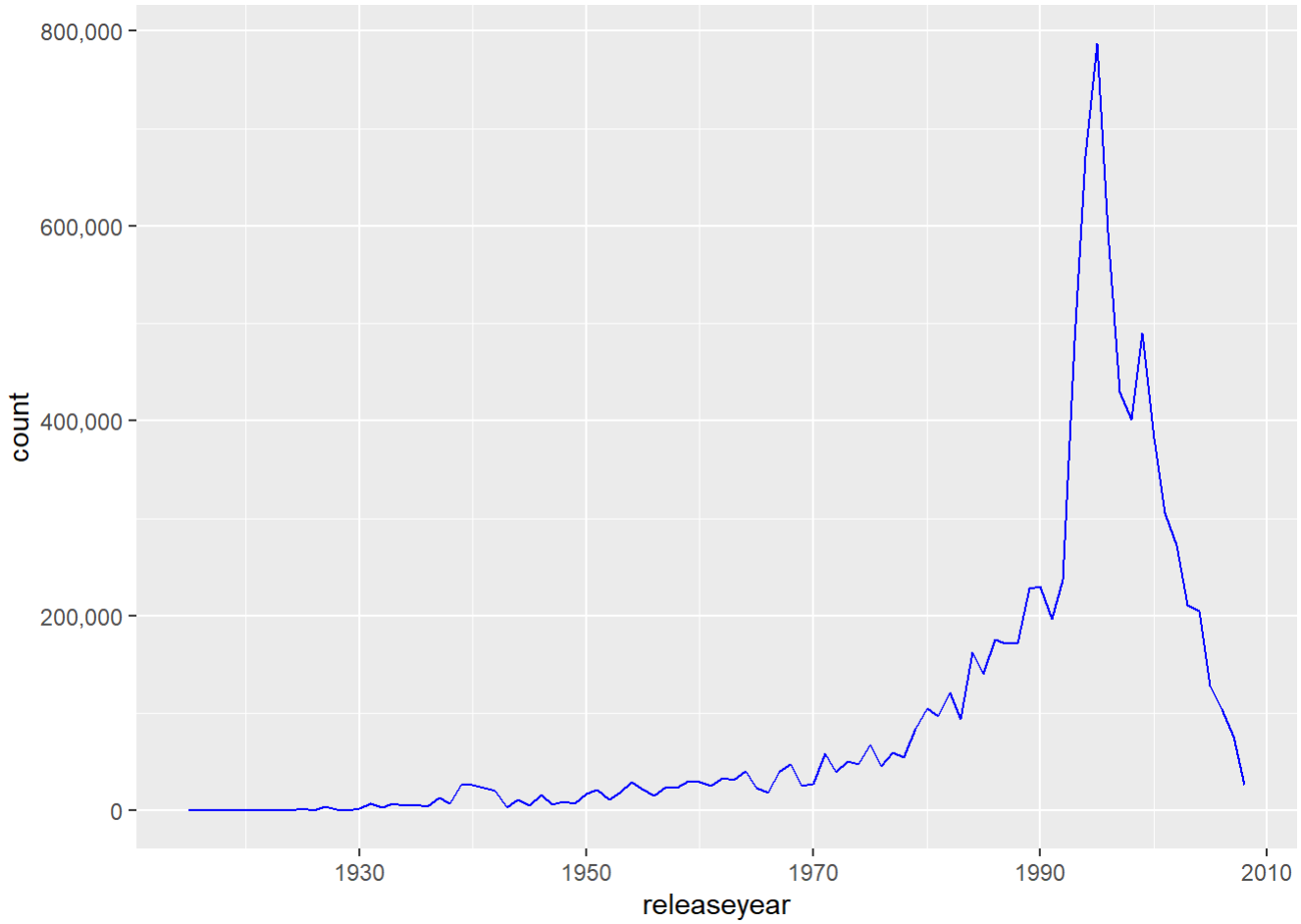
genres	count
Drama	733353
Comedy	700883
Comedy|Romance	365894
Comedy|Drama	323518
Comedy|Drama|Romance	261098
Drama|Romance	259735
Action|Adventure|Sci-Fi	220363
Action|Adventure|Thriller	148933
Drama|Thriller	145359
Crime|Drama	137424

genres	count
Drama|War	111122
Crime|Drama|Thriller	105893
Action|Adventure|Sci-Fi|Thriller	104906
Action|Crime|Thriller	102120
Action|Drama|War	99086
Action|Thriller	96235
Action|Sci-Fi|Thriller	95486
Thriller	94651
Horror|Thriller	75106
Comedy|Crime	73343

There was an exponential growth of the movie count especially in 1995 and a sudden drop in 2010. The reason for the drop was that the data was not collected until Oct 2009; therefore, we don't have the full data on this year.

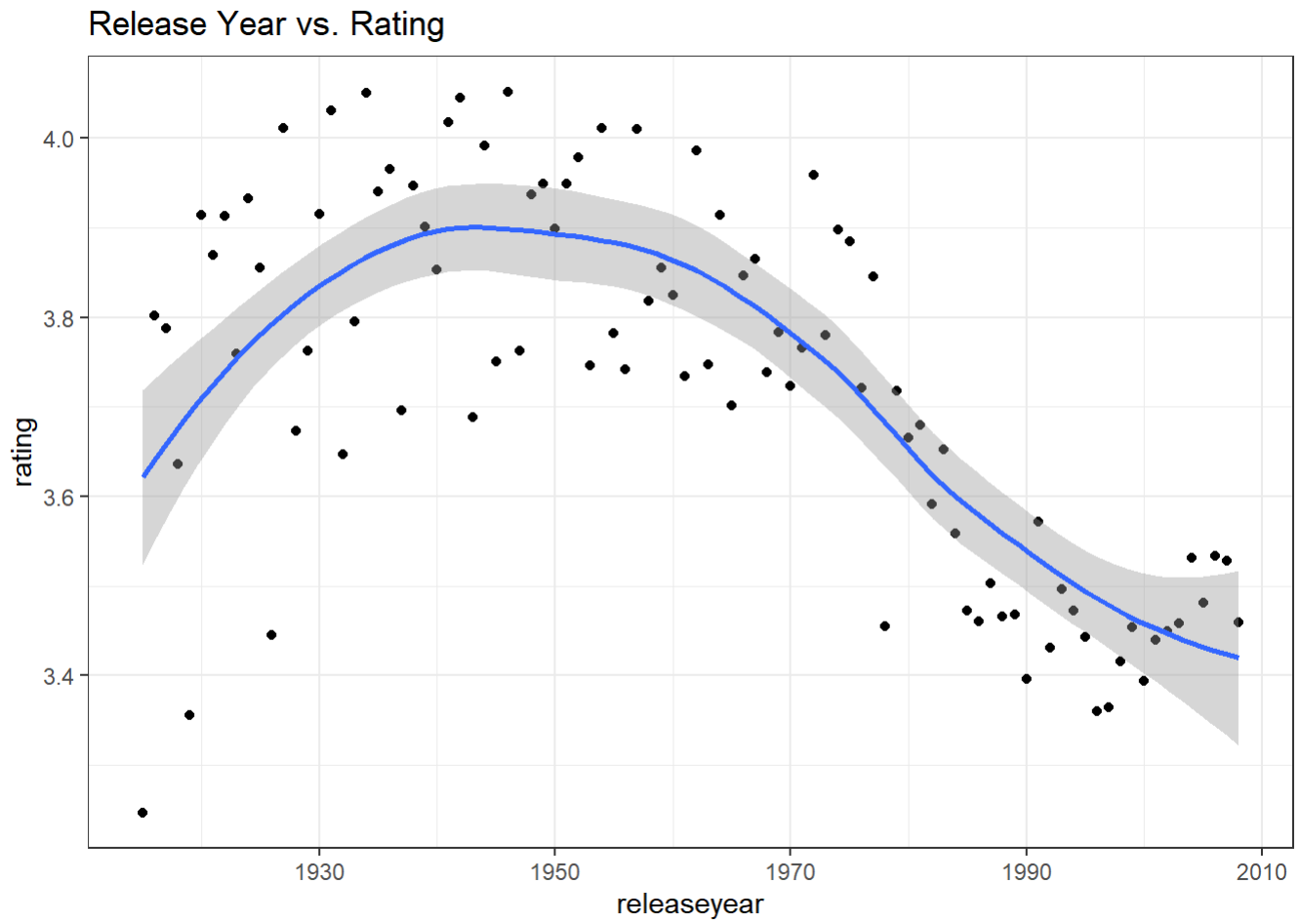
Note that the growth of the internet didn't start until 1985. There was barely any rating prior to 1985 due to lack of internet and public access.

```
# Exam the movie release years
# Extract release year from string title into a separate numeric field
  edx <- edx %>% mutate(releaseyear = as.numeric(str_extract(str_extract(title, "[/(]\\d{4}[/)]
$"), regex("\\d{4}"))),title = str_remove(title, "[/(]\\d{4}[/)]$"))

# Number of movies per year/decade
 movies_per_year <- edx %>%
    select(movieId, releaseyear) %>% # select columns we need
    group_by(releaseyear) %>%        # group by year
    summarise(count = n())  %>%      # count movies per year
    arrange(releaseyear)

# Exam the rating trend based on releaseyear
movies_per_year %>%
  ggplot(aes(x = releaseyear, y = count)) +
  scale_y_continuous(labels = comma) +
  geom_line(color="blue")
```



Older movies seem to get higher rating.

```
# Exam release year vs ratings mean
edx %>% group_by(releaseyear) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(releaseyear, rating)) +
  geom_point() +
  theme_hc() +
  geom_smooth() +
  theme_bw()+
  ggtitle("Release Year vs. Rating")
```



It would be difficult to factor genre into the overall prediction because certain genres being more popular in certain periods.

Data Wrangling

Partition edx dataset further into train and test sets 90/10 ratio respectively.

```
##############################################################################################
# Data Wrangling
##############################################################################################
 set.seed(1)  # This is a randomized algorithm
 test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
 train <- edx[-test_index,]
 temp <- edx[test_index,]

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

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

  rm(test_index, temp, removed)
```

Predictive Model Approach & Evaluation

Several models will be developed and assessed starting with the simplest. Accuracy will be evaluated using the RMSE.

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

N is defined as the number of user/movie combination,$Y_{u,i}$ as the rating for movie i by user u with the prediction as $\hat{Y}_{u,i}$. To compute RMSE, use the loss function to calculate the residual (difference between prediction and truth) for each data point (rating). Basically the loss function computes the RMSE as the measure of accuracy for the error in rating.

For this project, if the number is larger than 1 it means the typical error is larger than one star. The goal is to reduce the error below **0.8649.** Accuracy will be compared across all models using the Loss function below:

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

Model 1 - A Naive Model Approach

The first model is the simplest recommendation system by assuming the same rating (the average) for all movies regardless of user and all differences were assumed to be random variation around this “true” rate.

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

$\varepsilon_{u,i}$ as the independent sample errors and μ as the true rating for all movies.

```
 mu_hat <- mean(train$rating)
 mu_hat
```

```
## [1] 3.512354
```

In this case the lowest RMSE would be attained using the observed average of the data set as the estimate, with a resulting RMSE of:

```
mu_hat <- mean(train$rating)
naive_rmse <- RMSE(test$rating, mu_hat)
naive_rmse
```

```
## [1] 1.059
```

This is the first RMSE. Different approaches will be compared. Let's create a result table with this first RMSE.

Method	RMSE
Model 1 - Naive (Observed Average)	1.059

This typical error is greater than 1 star. It was our first simple attempt that lacks accuracy. Our next model will build on this.

Model 2 - A Movie Effects Approach

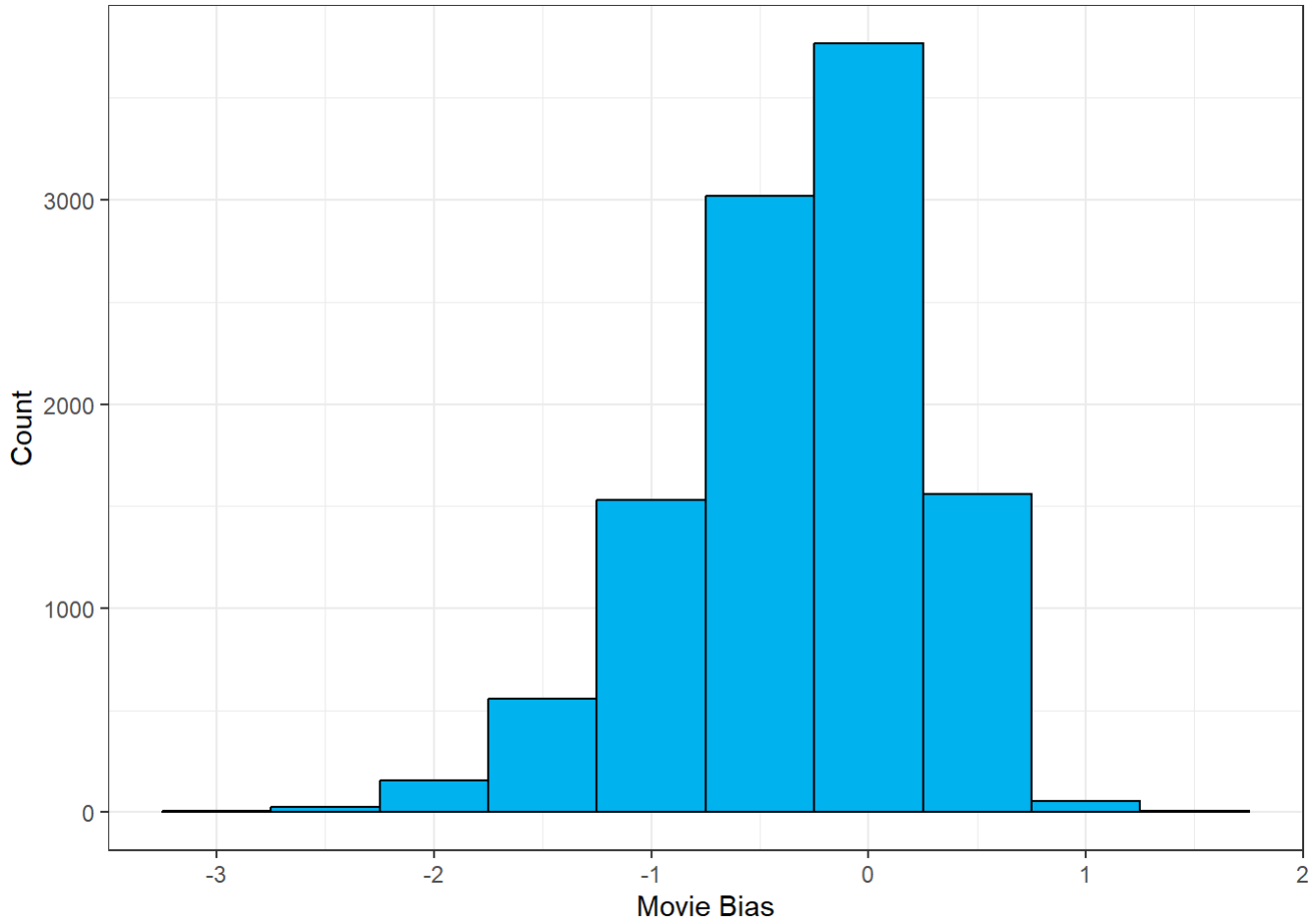
We know from experience and data confirms this, that some movies are more popular than others and receive higher ratings. Taking into account the bias effects associated with the movies, we can add the b_i to the existing model to reflect the bias.

b_i represents the average effect for movie i:

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

This distribution shows the bias. The mean is at 0 so a b_i of 1.5 reflects a 5 star rating.

```
mu <- mean(train$rating)

# bi is the movies averages
bi <- train %>%
   group_by(movieId) %>%
   summarize(bi = mean(rating - mu))

bi %>% ggplot(aes(bi)) +
 geom_histogram(color = "black", fill = "deepskyblue2", bins = 10) +
 xlab("Movie Bias") +
 ylab("Count") +
 theme_bw()
```



When we account for the movie effect in the model, the RMSE is reduced.

```
predicted_ratings <- mu + test %>%
   left_join(bi, by='movieId') %>%
   pull(bi)

 bi_rmse <- RMSE(predicted_ratings, test$rating)
 bi_rmse
```

```
## [1] 0.9426564
```

Method	RMSE
Model 1 - Naive (Observed Average)	1.0590002
Model 2 - Movie Effects	0.9426564

Model 3 - A User Effects Approach

There is substantial variability across users as well.Some users are very cranky and tend to rate negatively and others love every movie and rate more positively. Taking into account the bias effects associated with the user, we can add the b_u to the existing model to reflect the bias.

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

When we account for both the movie and user effects in the model, the RMSE is reduced further.

```
#bu is the user averages.
bu <- train %>%
    left_join(bi, by="movieId") %>%
    group_by(userId) %>%
    summarize(bu = mean(rating - mu - bi))


  predicted_ratings <- test %>%
    left_join(bi, by='movieId') %>%
    left_join(bu, by='userId') %>%
    mutate(pred = mu + bi + bu) %>%
    pull(pred)

  bu_rmse <-  RMSE(predicted_ratings, test$rating)
  bu_rmse
```

```
## [1] 0.8646047
```

Method	RMSE
Model 1 - Naive (Observed Average)	1.0590002
Model 2 - Movie Effects	0.9426564
Model 3 - User Effects	0.8646047

Model 4 - Regularization

Some of the data is noisy. For example ratings on obscure or niche movies by only a few users. This adds variability and can increase RMSE. We can use regularization to avoid over fitting and penalize large estimates formed by small sample sizes to reduce this effect. The optimal value of λ is determined through cross validation and applied to our model.

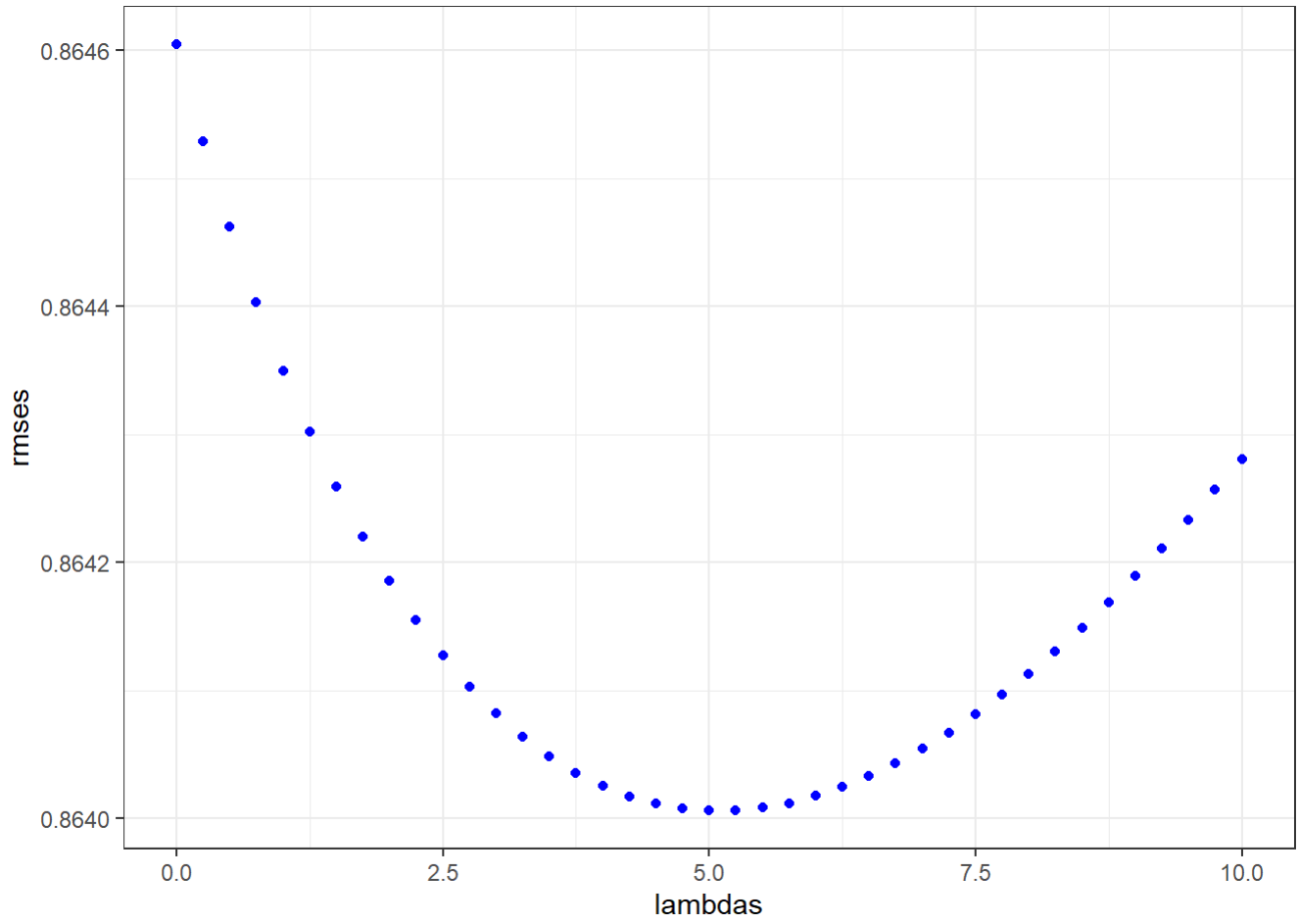
$$\sum_{u,i} (y_{u,i} - \mu - b_i - b_u)^2 + \lambda \left(\sum_i b_i^2 + \sum_u b_u^2 \right)$$

This code displays the RMSE associated with various values of λ.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train$rating)
  bi <- train %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(n()+l))
  bu <- train %>%
    left_join(bi, by="movieId") %>%
    group_by(userId) |>
    summarize(bu = sum(rating - bi - mu)/(n()+l))
  predicted_ratings <-
    test %>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    mutate(pred = mu + bi + bu) %>%
    pull(pred)
    return(RMSE(predicted_ratings, test$rating))
})
```

This Graph shows a range of lambdas VS RMSE. The optimal setting provides the lowest error.

```
qplot(lambdas, rmses, color=I("blue"))+
theme_bw()
```



The value of lambda which results in lowest RMSE is:

```
  lambda <- lambdas[which.min(rmses)]
  lambda
```

```
## [1] 5
```

The RMSE has improved however it is a very small gain in accuracy.

Method	RMSE
Model 1 - Naive (Observed Average)	1.0590002
Model 2 - Movie Effects	0.9426564
Model 3 - User Effects	0.8646047
Mdoel 4 - Regularized Movie + User Effects	0.8640060

Let’s try another approach.

Model 5 Recommender Systems

Recommender systems use historical data to make predictions. It is based on historical behavior by its users. So far we have approached a dataset that features sparsity and biases with models that account for these effects with decent accuracy. To get better results we turn to a more advanced method called matrix factorization. Our user data is processed as a large and sparse matrix, then decomposed into two smaller dimensional matrices with latent features and less sparsity. To make the process more efficient the recosystem packgage will be used. For more information on Recommender System Using Parallel Matrix Factorization (https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html). We start by converting data into the recosystem format, finding the best tuning parameters, training,and finally testing it.

Be advised this process may take from 4 to more than 30 minutes to run.

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

  library(recosystem)
  # data_memory(): Specifies a data set from R objects
  set.seed(1, sample.kind="Rounding")
  train_reco <- with(train, data_memory(user_index = userId, item_index = movieId, rating = rati
ng))
  test_reco <- with(test, data_memory(user_index = userId, item_index = movieId, rating = ratin
g))

  # Create a model object (a Reference Class object in R) by calling Reco().
  r <- Reco()

  # select best tuning parameters along a set of candidate values
  para_reco <- r$tune(train_reco, opts = list(dim = c(20, 30),
                                              costp_l2 = c(0.01, 0.1),
                                              costq_l2 = c(0.01, 0.1),
                                              lrate = c(0.01, 0.1),
                                              nthread = 4,
                                              niter = 10))

  # Train the model
  r$train(train_reco, opts = c(para_reco$min, nthread = 4, niter = 30))
```

```
## iter      tr_rmse          obj
##    0       0.9812   1.1005e+07
##    1       0.8761   8.9994e+06
##    2       0.8429   8.3516e+06
##    3       0.8200   7.9542e+06
##    4       0.8037   7.6855e+06
##    5       0.7915   7.4959e+06
##    6       0.7813   7.3527e+06
##    7       0.7727   7.2349e+06
##    8       0.7653   7.1389e+06
##    9       0.7590   7.0599e+06
##   10       0.7537   6.9977e+06
##   11       0.7488   6.9401e+06
##   12       0.7444   6.8901e+06
##   13       0.7404   6.8486e+06
##   14       0.7369   6.8105e+06
##   15       0.7335   6.7775e+06
##   16       0.7304   6.7470e+06
##   17       0.7276   6.7207e+06
##   18       0.7249   6.6946e+06
##   19       0.7224   6.6703e+06
##   20       0.7202   6.6506e+06
##   21       0.7180   6.6314e+06
##   22       0.7161   6.6144e+06
##   23       0.7141   6.5981e+06
##   24       0.7124   6.5829e+06
##   25       0.7107   6.5677e+06
##   26       0.7092   6.5561e+06
##   27       0.7077   6.5443e+06
##   28       0.7063   6.5318e+06
##   29       0.7050   6.5201e+06
```

```
  # Compute predicted values
  results_reco <- r$predict(test_reco, out_memory())
```

With the algorithm trained now we can test to see the resulting RMSE.

```
factorization_rmse <- RMSE(results_reco, test$rating)
```

Method	RMSE
Model 1 - Naive (Observed Average)	1.0590002
Model 2 - Movie Effects	0.9426564
Model 3 - User Effects	0.8646047
Mdoel 4 - Regularized Movie + User Effects	0.8640060
Model 5: Matrix Factorization Using Recosystem	0.7847194

This is a great improvement. The RMSE is significantly less than the target RMSE. **We have our model.**

Final Validation

Now that we've found the lowest RMSE, the final step is to use matrix factorization to train it using the edx dataset and then test its accuracy on the validation set.

Be advised this process may take from 20 to more than an hour to run.

```
set.seed(1, sample.kind="Rounding")
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, ratin
g = rating))

r <- Reco()

#Be advised, this may take more than 45 min to run:
para_reco <- r$tune(edx_reco, opts = list(dim = c(20, 30),
                                          costp_l2 = c(0.01, 0.1),
                                          costq_l2 = c(0.01, 0.1),
                                          lrate = c(0.01, 0.1),
                                          nthread = 4,
                                          niter = 10))

r$train(edx_reco, opts = c(para_reco$min, nthread = 4, niter = 30))
```

```
## iter      tr_rmse          obj
##    0       0.9731   1.1992e+07
##    1       0.8739   9.8994e+06
##    2       0.8400   9.1876e+06
##    3       0.8171   8.7558e+06
##    4       0.8013   8.4796e+06
##    5       0.7894   8.2771e+06
##    6       0.7796   8.1230e+06
##    7       0.7715   8.0036e+06
##    8       0.7646   7.9051e+06
##    9       0.7586   7.8242e+06
##   10       0.7535   7.7562e+06
##   11       0.7488   7.6995e+06
##   12       0.7448   7.6491e+06
##   13       0.7410   7.6060e+06
##   14       0.7377   7.5680e+06
##   15       0.7347   7.5334e+06
##   16       0.7318   7.5031e+06
##   17       0.7293   7.4764e+06
##   18       0.7267   7.4504e+06
##   19       0.7246   7.4274e+06
##   20       0.7225   7.4081e+06
##   21       0.7205   7.3894e+06
##   22       0.7186   7.3695e+06
##   23       0.7170   7.3542e+06
##   24       0.7153   7.3397e+06
##   25       0.7138   7.3242e+06
##   26       0.7123   7.3121e+06
##   27       0.7111   7.3005e+06
##   28       0.7098   7.2895e+06
##   29       0.7086   7.2793e+06
```

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

```
final_rmse <- RMSE(final_reco, validation$rating)
```

Method	RMSE
Model 1 - Naive (Observed Average)	1.0590002
Model 2 - Movie Effects	0.9426564
Model 3 - User Effects	0.8646047
Mdoel 4 - Regularized Movie + User Effects	0.8640060
Model 5: Matrix Factorization Using Recosystem	0.7847194
Final Validation: Matrix factorization using recosystem	0.7820791

Conclusion

The final RMSE is 0.7805. Significantly below the target of 0.8649. We developed and tested several models and achieved the best accuracy using matrix factorization which was simplified through the recosystem package.

References

1. Introduction to Data Science, Rafael A. Irizarry, Recommendation systems, last updated 2022, 12, 05, http://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#user-effects

2. Winning the Netflix Prize: A Summary, Edwin Chen, http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/

3. 10M MovieLensDataset Version, http://files.grouplens.org/datasets/movielens/ml-10m.zip

3. 2023 GroupLens, MovieLens 10M Dataset,https://grouplens.org/datasets/movielens/10m/

4. Recommender System Using Parallel Matrix Factorization, https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html

5. How to Create an R Markdown Research Report, Carsten Lange, https://www.youtube.com/watch?v=agFAR_EmXtw