HarvardX: PH125.9X - Capstone MovieLens Project

Arnab K Sarkar

2024-12-10

Contents

1 Overview

This project is part of the "HarvardX: Data Science PH125.9x Capstone MovieLens" course, inspired by the MovieLens dataset and aims to develop a predictive model for movie ratings. The report begins by outlining the project's objective and methodology, followed by preparing and exploring the dataset to uncover key insights. Leveraging these insights, a machine learning algorithm is designed and evaluated to predict movie ratings with accuracy. The results of the analysis are presented in detail, culminating in concluding observations and reflections on the findings.

1.1 Introduction

Recommendation systems have revolutionized the way users interact with products and services by leveraging user-generated ratings to make personalized suggestions. Companies like Amazon use vast amounts of customer rating data to predict preferences and recommend items tailored to individual users, enhancing their shopping experience and driving engagement. By predicting high ratings for certain items based on past behavior, recommendation systems help users discover products they are most likely to appreciate.

This concept extends beyond retail to entertainment, as seen in the case of movies. Recommendation systems are among the most widely applied machine learning models, playing a pivotal role in platforms like Netflix, whose success is largely attributed to its robust recommendation engine. In 2009, Netflix awarded a $1M prize to the team of data scientists who had successfully met the challenge of improving their movie recommendation algorithm by 10% (Lohr 2009; Koren 2009). That highlights the importance of recommendation systems in driving user satisfaction and business success.

In this project, we focus on building a movie recommendation system using the 10M MovieLens dataset, collected by GroupLens Research. The MovieLens datasets have provided a popular environment for experimentation with machine learning since their launch in 1997 (Harper and Konstan 2015).

1.2 Project Objective

The objective of this project is to develop a machine learning algorithm capable of predicting user ratings for movies on a scale of 0.5 to 5 stars. The model will be trained using the provided edx dataset and evaluated on its ability to accurately predict ratings in the final holdout dataset.

The performance of the algorithm will be assessed using the Root Mean Square Error (RMSE), a widely used metric to measure the accuracy of predictions. RMSE evaluates the differences between predicted and observed values, with lower RMSE values indicating better model performance. Since RMSE is sensitive to larger errors due to its squared-error calculation, minimizing RMSE ensures the model's robustness to outliers.

To achieve the objective, five predictive models will be developed, each evaluated and compared based on their respective RMSE values, followed by applying a regularization principle in the end. The target for the algorithm is to achieve an RMSE lower than 0.86490.

The mathematical formulation for RMSE is as follows:

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

Where $y_{u,i}$ = the actual rating provided by user i for movie u \ $\hat{y}_{u,i}$ = the predicted rating for the same \ N = the total number of user/movie combinations.

Finally, the model with the best RMSE performance will be applied to predict movie ratings in the validation dataset, demonstrating the effectiveness of the developed recommendation system.

```r
##########################################################
# Create edx and final_holdout_test sets
##########################################################

# Note: this process could take a couple of minutes

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

library(tidyverse)
library(caret)

# 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")
```

To achieve the best accurate predictions for user ratings of movies they have not yet seen, the MovieLens dataset will be split into two subsets:

Training Subset (edx): This subset will be used to train the machine learning algorithm. It contains the majority of the data, enabling the model to learn patterns and relationships between users, movies, and ratings.

Final Data Set (final_holdout_test): This subset will be used to evaluate the performance of the trained algorithm. It serves as an independent test set to measure how well the model predicts ratings for unseen data.

```
# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
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)
edx <- rbind(edx, removed)

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

```
#Save edx and validation files
save(edx, file="edx.RData")
save(final_holdout_test, file = "final_holdout_test.RData")
```

```
#Install Additional packages as needed.

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

2 Exploratory Analysis

The edx dataset is a data.frame consisting of 9,000,055 rows and 6 columns, with ratings provided by a total of 69,878 unique users for a total of 10,677 unique movies. If each unique user had provided a rating for each unique rating the dataset would include a total of approximately 746 million ratings. Clearly, therefore, this dataset includes many missing values, i.e. every user has not rated every movie.

```r
# explore dataset
str(edx)
```

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

```r
str(final_holdout_test)
```

```
## 'data.frame':    999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : int  231 480 586 151 858 1544 590 4995 34 432 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Romanc
```

```r
# Tabulate first 5 rows included in edx dataset
rbind((lapply(edx, class)), head(edx)) %>%
  kable(caption = "edx dataset: first 5 rows with variable type", align = 'cccllll', booktabs = T,
        format = "latex", linesep = "") %>%
  row_spec(1, hline_after = T) %>%
  kable_styling(full_width = FALSE, position = "center",
                latex_options = c("scale_down", "hold_position"))
```

Table 1: edx dataset: first 5 rows with variable type

	userId	movieId	rating	timestamp	title	genres
1	integer	integer	numeric	integer	character	character
11	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|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

```
# Confirm if there are any blank/NA values in the dataset.
  colSums(is.na(edx))
```

```
##    userId   movieId    rating timestamp     title    genres
##         0         0         0         0         0         0
```

A Column summary of the subset confirms that there are no missing values.

```
##   unique_users unique_movies
## 1        69878         10677
```

2.1 Ratings Distribution

The overall average rating in the edx dataset was 3.51. The minimum rating awarded to any movie was 0.5 and the maximum rating awarded was 5. The distribution of total ratings included in the dataset (Figure 1) shows that the most common rating across all movies was 4, and that, overall, whole star ratings (7,156,885; 79.5%) were used more than half star ratings (1,843,170; 20.5%). Users have a preference to rate movies rather higher than lower as shown by the distribution of ratings below. 4 is the most common rating, followed by 3 and 5. 0.5 is the least common rating. In general, half rating are less common than whole star ratings.

```
# Plot distribution of ratings in the edx dataset
# Ratings distribution
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.25, color = "black" , fill= "light blue") +
  scale_x_continuous(breaks = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 1000000))) +
  ggtitle("Distribution of Movie Ratings") + plot_theme
```

2.2 Movie distribution by Rating

Some movies have been rated more often than others, while some have very few ratings and sometimes only one rating. Here are the count of Movies that are rated only once and subsequently removed from the edX dataset to improve the reliability of the analysis.

```
## [1] 126
```

```
# Plot number of ratings per movie
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 25, color = "black" , fill= "light blue") +
  scale_x_log10() +
  xlab("# of Ratings") +
  ylab("# of Movies") +
  ggtitle("Movie distribution by average rating")
```

There is clearly a movie effect on the rating awarded and, as such, adjusting for this effect (or bias) was considered worthwhile for inclusion in the training algorithm.

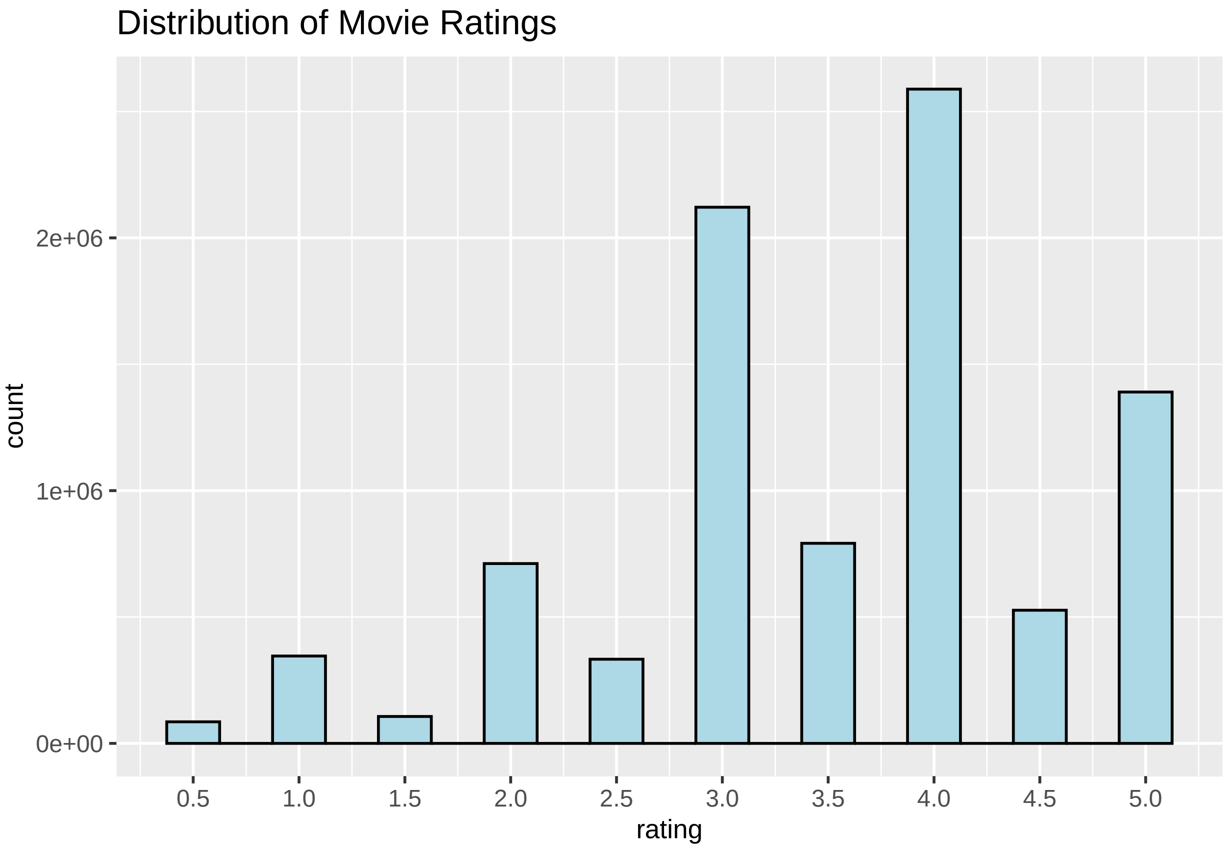


Figure 1: Overall ratings distribution

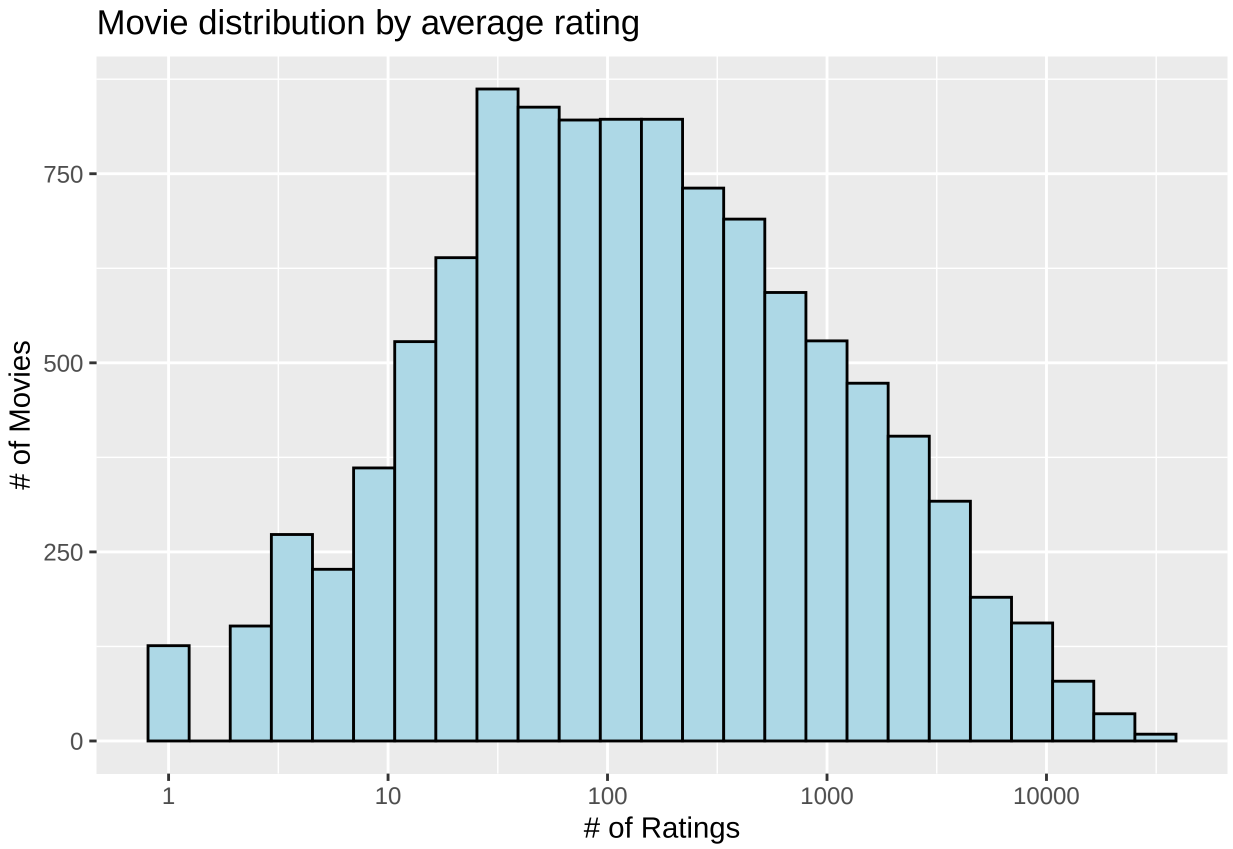


Figure 2: Movie distribution by average rating

2.3 User effect

Exploration of user data revealed a similar pattern to that observed for movies, with some users appearing more generous in the way they assessed movies, having provided higher ratings than others (see Fig. "User distribution by average rating"). Some users contributed many more ratings than other users. For example, one user provided a total of 6616 ratings whereas as many as 1059 provided fewer than 10 movie ratings each. This analysis identifies a clear user effect (or bias) which, if adjusted for, may further improve the accuracy of a movie recommendation system.

```
# Plot count of ratings by users
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 20, color = "black", fill= "light blue") +
  scale_x_log10() +
  xlab("# of ratings") +
  ylab("# of users") +
  ggtitle("Count of Ratings by Users")
```

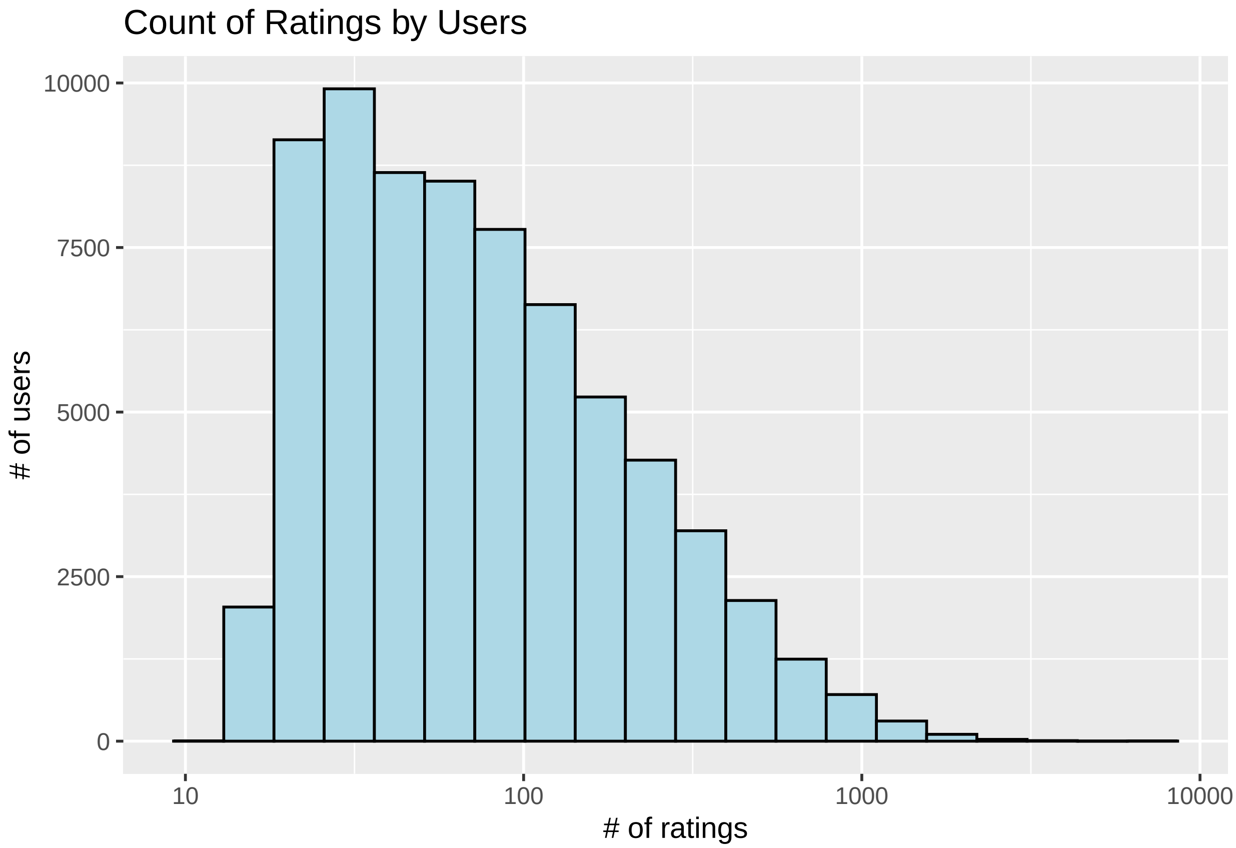


Figure 3: User distribution by average rating

The data reveals that most users have provided ratings for between 30 and 100 movies. This suggests the need to include a user-specific penalty term in our models to account for individual differences in rating behavior. Moreover, users exhibit considerable variability in their rating tendencies. While some users consistently give lower star ratings, others are inclined to provide higher-than-average ratings. The visualization below focuses on users who have rated at least 100 movies, providing a clearer picture of these differences.

```
# Plot mean movie ratings by users
edx %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 20, color = "black", fill= "light blue") +
  scale_x_continuous(breaks = c(seq(1,5,0.5))) +
  xlab("Mean Rating") +
  ylab("# of Users") +
  ggtitle("Mean Ratings by Users")
```

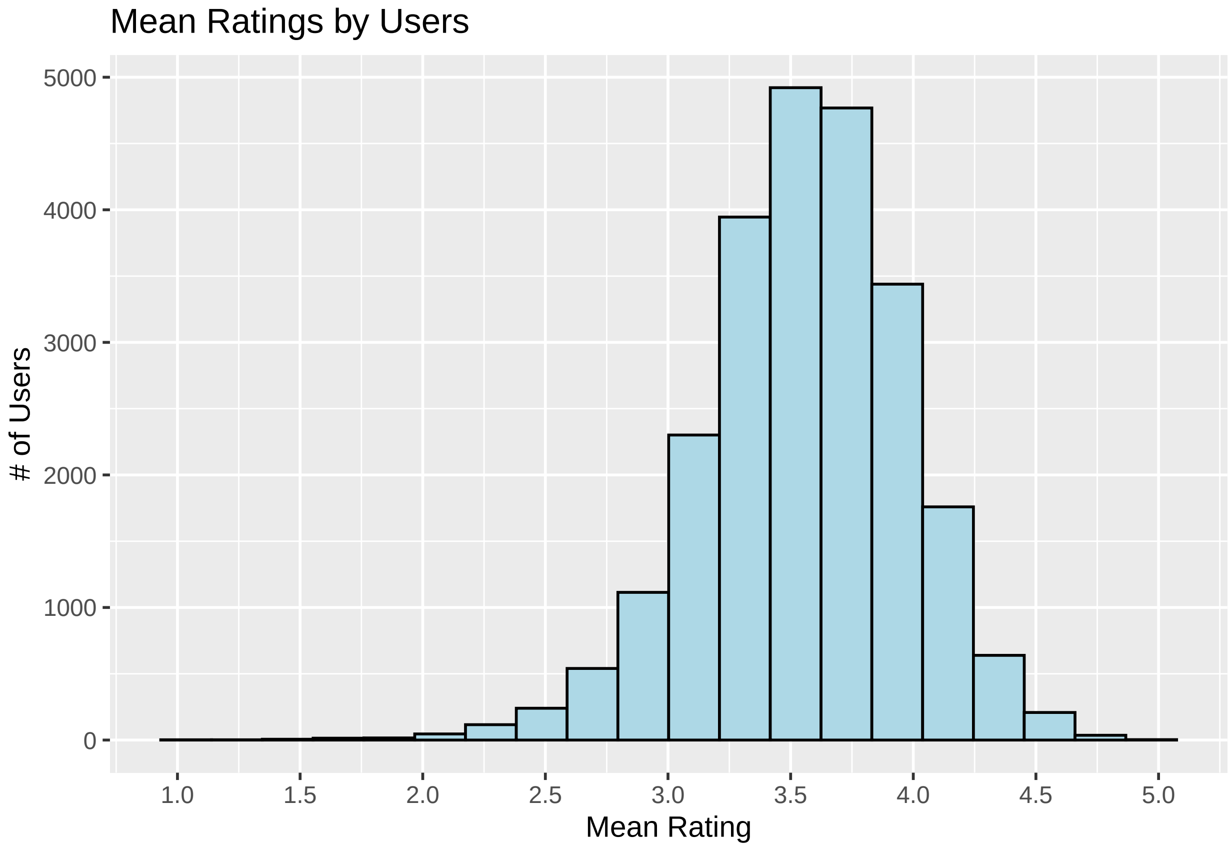


Figure 4: Mean ratings by user

2.4 Movie Genre

As shown in Table 1, the genres variable provides the genre labels associated with each movie. For example, the movie r edx$title[1]fromthefirstrowoftheedxdatasetiscategorizedundertheredx$genres[1] genre(s). Many movies belong to multiple genres, leading to a total of r n_distinct(edx$genres) unique genre combinations across the dataset. By separating these combinations into individual rows for single genres, we identified r edx %>% separate_rows(genres, sep = "\|") %>% group_by(genres) %>% count(genres) %>% nrow() distinct genre categories, including entries for "no genre listed."

Table 2 highlights the distribution of ratings across these categories. Drama and comedy movies received the highest number of ratings, while genres such as Documentary and IMAX had the fewest. Notably, there were only seven ratings for movies with no genre specified. The table also reveals variations in average ratings across genres.

For visualization, the data was grouped by unique genre combinations, focusing on combinations with at least 100,000 ratings to simplify the analysis. This filtered view underscores a noticeable genre effect. Movies categorized as 'Comedy' had the lowest average ratings, while genres like 'Crime|Drama' and 'Drama|War' achieved the highest averages (Figure "Average rating by genre"). This clear genre-dependent variation emphasizes the importance of incorporating genre effects when training the recommendation system algorithm.

```r
# Separate individual genres and ranking them by the total number of ratings in the edx dataset
edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(count = n(), rating = round(mean(rating), 2)) %>%
  arrange(desc(count)) %>%
  kable(col.names = c("Genre", "No. of Ratings", "Avg Rating"),
        caption = "Individual genres ranked by count of ratings",
        align = "lrr", booktabs = TRUE, format = "latex", linesep = "") %>%
  kable_styling(full_width = FALSE, position = "center", latex_options = "hold_position")
```

Table 2: Individual genres ranked by count of ratings

Genre	No. of Ratings	Avg Rating
Drama	3910127	3.67
Comedy	3540930	3.44
Action	2560545	3.42
Thriller	2325899	3.51
Adventure	1908892	3.49
Romance	1712100	3.55
Sci-Fi	1341183	3.40
Crime	1327715	3.67
Fantasy	925637	3.50
Children	737994	3.42
Horror	691485	3.27
Mystery	568332	3.68
War	511147	3.78
Animation	467168	3.60
Musical	433080	3.56
Western	189394	3.56
Film-Noir	118541	4.01
Documentary	93066	3.78
IMAX	8181	3.77
(no genres listed)	7	3.64

```r
# Plot average rating by genre for genre combinations with at least 100,000 ratings
edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 100000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(x = "Genre combination", y = "Average Rating") + plot_theme
```

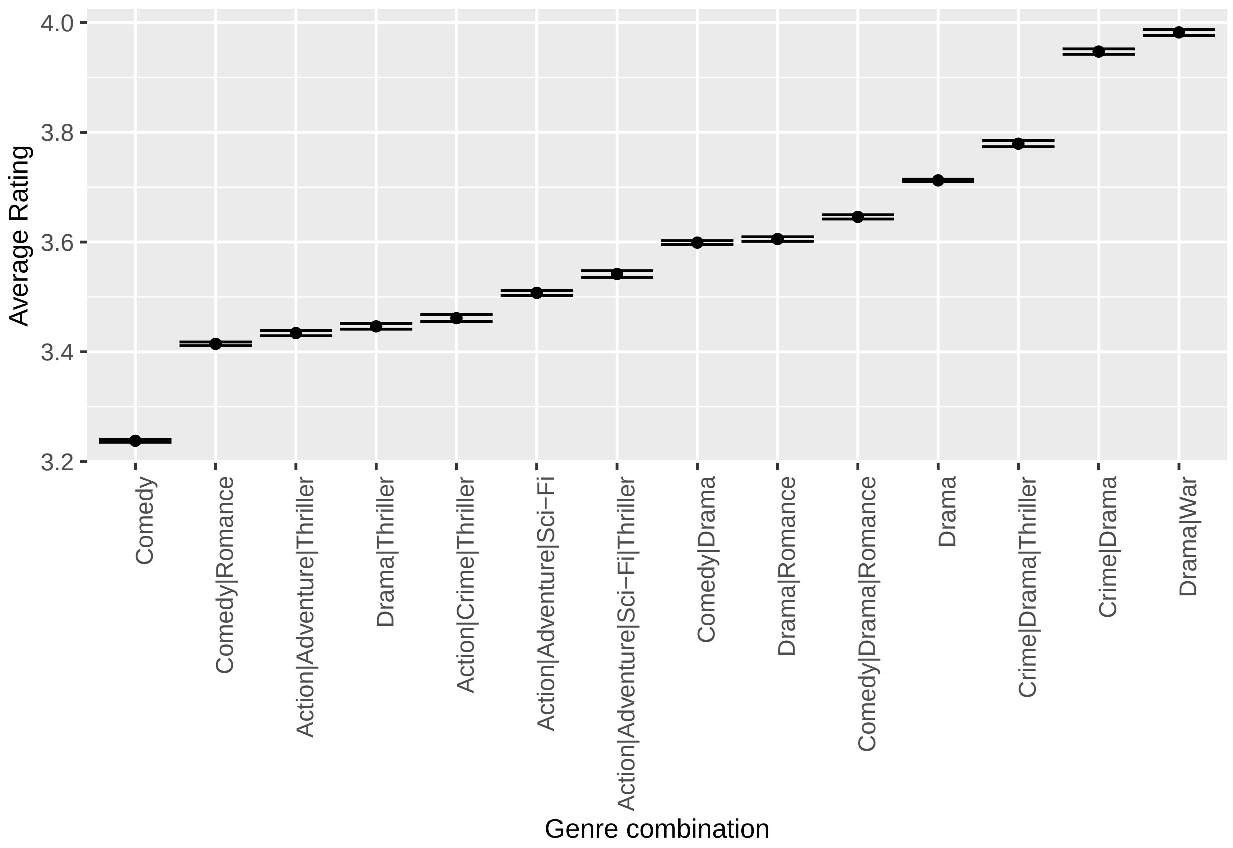


Figure 5: Average rating by genre

2.5 Date of Review

To analyze the effect of review dates on ratings, the timestamp data was converted into a date format, excluding time details and rounding to the nearest week. This approach helped smooth fluctuations and provided a clearer trend for the analysis.

```r
#Add Review Year column to the dataset
edx <- edx %>%
    mutate(review_date = round_date(as_datetime(timestamp), unit = "week"))
```

The earliest review in the dataset dates back to 1995, a year marked by the highest observed average rating. From this point, a gradual decline in average ratings was noted until around 2005, after which ratings began to rise again. While the impact of review date on average ratings was less pronounced compared to the effects of movies and users, there was still noticeable variation over time (see Fig. "Average rating by review date"). This supports the inclusion of review date as a factor in the development of the recommendation algorithm.

```r
# Plot average rating by date of review in the edx dataset
edx %>% group_by(review_date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(review_date, rating)) +
  geom_point() +
  geom_smooth() +
  xlab("Year of Review") +
  ylab("Average Rating") +
  ggtitle("Average Rating by Review Date") +
  theme_light()
```

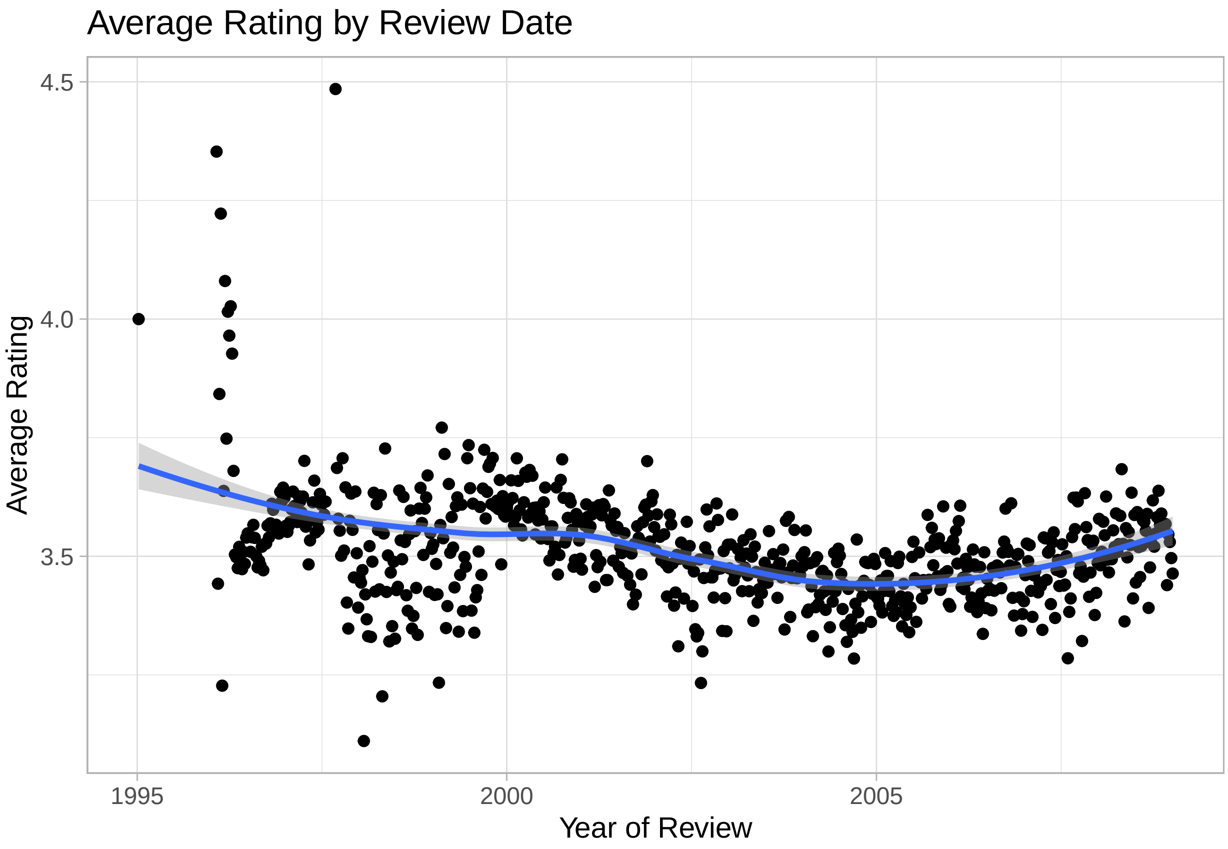


Figure 6: Average rating by review date

3 Methods

3.1 Split edx data set into train and test sets

As the validation dataset was reserved for the final hold-out test, the edx dataset needed to be used both to train and test the algorithm in development. This is important to allow for cross-validation and refinement of the final model without the risk of over-training. Other methods for cross-validation include K-fold cross validation and bootstrapping but were not utilized here.

Here, the same technique was applied as with the original movielens dataset, using the caret function ‘createDataPartition’ to divide the edx dataset into train (80%) and test (20%) sets. As before, the dplyr functions ‘semi_join’ and ‘anti_join’ were used, firstly to ensure that the test set only included users and movies that are present in the train set and, secondly to add the removed data to the train set in order to maximize the data available for training purposes.

```r
##############################################################
# Methods - partition edx into train and test sets
##############################################################

set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_temp_set <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set

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

# add the removed data to the train set using the anti_join function:
removed_set <- anti_join(test_temp_set, test_set)
```

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

```r
train_set <- rbind(train_set, removed_set)

# Remove temporary files to tidy environment
rm(test_index, test_temp_set, removed_set)
```

3.2 Develop the Model

The objective of the project was to develop an algorithm that achieved an RMSE below 0.86490 as set out below. A simple table was created to capture the project objective as well as the results obtained during development within the edx dataset and in the final hold-out test in the validation dataset (see Section 4: Results).

Method	RMSE
Target RMSE	0.8649

3.2.1 Simple Average Model

The first basic model predicts the same rating for all movies, so we compute the mean rating of the dataset. The expected rating of the underlying data set is between 3 and 4. We start by building the simplest possible model by predicting the same rating for all movies regardless of user who give it. A model based approach assumes the same rating for all movie with all differences explained by random variation :

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

with $\epsilon_{u,i}$ independent error sample from the same distribution centered at 0 and μ the "true" rating for all movies. This very simple model makes the assumption that all differences in movie ratings are explained by random variation alone. We know that the estimate that minimize the RMSE is the least square estimate of $Y_{u,i}$, in this case, is the average of all ratings: The expected rating of the underlying data set is between 3 and 4.

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

```
mu_simple_avg <- mean(train_set$rating)
mu_simple_avg
```

```
## [1] 3.512478
```

```
simple_avg_rmse <- RMSE(test_set$rating, mu_simple_avg)
simple_avg_rmse
```

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

Method	RMSE
Target RMSE	0.864900
Simple Average Model	1.059904

3.2.2 Movie effect model

To improve above model we focus on the fact that, from experience, we know that some movies are just generally rated higher than others. Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. We compute the estimated deviation of each movies' mean rating from the total mean of all movies μ. The resulting variable is called "b" (as bias) for each movie "i" b_i, that represents average ranking for movie i:

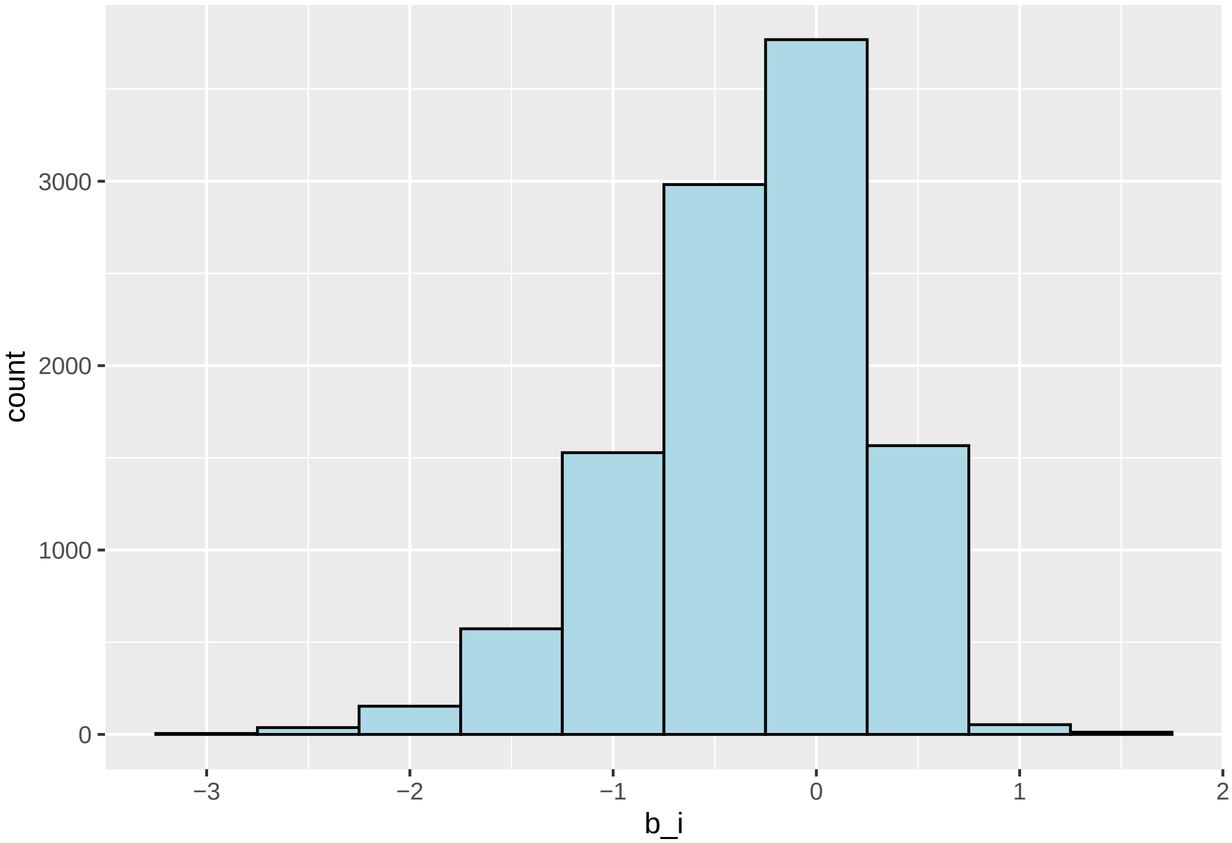
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

The histogram is left skewed, implying that more movies have negative effects.

```
#Exploratory analysis reveals that count of rating is not same across all movies.
#Some movies are rated more (or less) than others.
#Augment previous model by adding the term  b_i

mu_movie_avg <- mean(train_set$rating)
movie_avg <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_movie_avg))
movie_avg
```

```
## # A tibble: 10,677 x 2
##    movieId     b_i
##      <int>   <dbl>
##  1       1  0.418
##  2       2 -0.308
##  3       3 -0.371
##  4       4 -0.645
##  5       5 -0.448
##  6       6  0.303
##  7       7 -0.147
##  8       8 -0.394
##  9       9 -0.517
## 10      10 -0.0836
## # i 10,667 more rows
```

```
#plot variability in the estimate for movie effect
#movie_avg %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = "black", fill= "light blue")
movie_avg %>% ggplot(aes(b_i)) +
  geom_histogram(bins = 10, color = "black", fill= "light blue")
```



Our prediction improve once we predict using this model.

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

Method	RMSE
Target RMSE	0.8649000
Simple Average Model	1.0599043
Movie Effect Model	0.9437429

So we have predicted movie rating based on the fact that movies are rated differently by adding the computed b_i to μ. If an individual movie is on average rated worse that the average rating of all movies μ , we predict that it will rated lower that μ by b_i, the difference of the individual movie average from the total average. This approach shows improvement over the simple average model, as it accounts for variations in how individual movies are rated. However, it does not yet consider the effect of individual user rating behavior. Users differ in their tendencies to rate movies more critically or generously, which is an important factor to include in a more comprehensive model.

3.2.3 Movie and User effect model

The exploratory analysis revealed that different users exhibit distinct rating behaviors, with some being more critical and others more generous in their assessments. To account for this variability, further refinements were made to the algorithm by introducing user effects (b_u). As with the movie effects, rather than relying on linear regression models, the least square estimates of the user effect, $\hat{b}_u$ were computed. These estimates were derived using the formulas shown below, ensuring that the model adjusts predictions based on individual user tendencies. This refinement helps improve the accuracy of the predictions by incorporating personalized adjustments for each user.

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

$$\hat{b}_u = mean\left(\hat{y}_{u,i} - \hat{\mu} - \hat{b}_i\right)$$

We compute the average rating for user μ, for those that have rated over 50 movies.

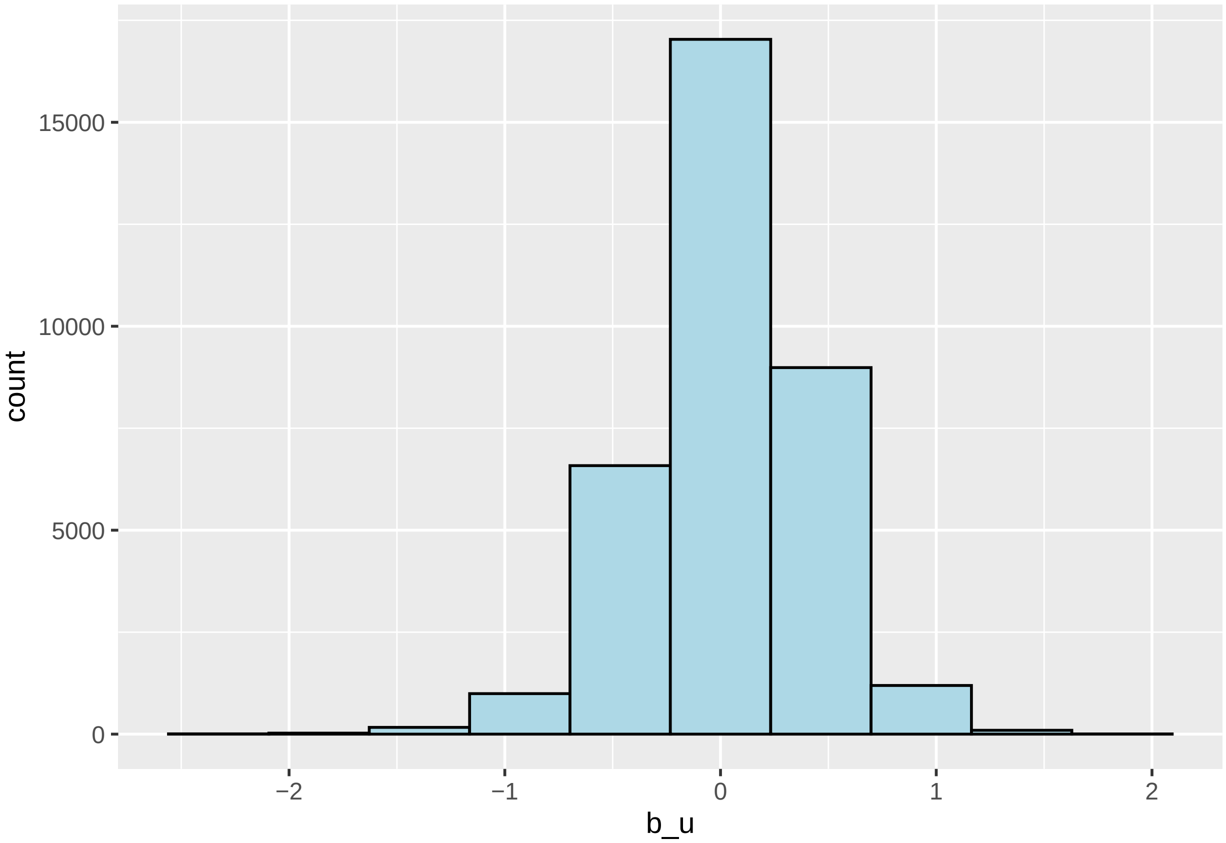
```r
mu_user_avg <- mean(train_set$rating)
mu_user_avg
```

```
## [1] 3.512478
```

```r
# Estimate user effect (b_u)
user_avg <- train_set %>%
  left_join(movie_avg, by="movieId") %>%
  group_by(userId) %>%
  filter(n()>=50) %>%
  summarize(b_u = mean(rating - mu_user_avg - b_i,na.rm = TRUE),
            .groups = "drop") #ensure proper grouping behavior

user_avg
```

```
## # A tibble: 35,087 x 2
##    userId     b_u
##     <int>   <dbl>
##  1      5  0.245
##  2      7 -0.0166
##  3      8  0.201
##  4     10  0.0800
##  5     13  0.0145
##  6     14 -0.129
##  7     18  0.108
##  8     19  0.0541
##  9     30  0.882
## 10     34 -0.473
## # i 35,077 more rows
```

```
# plot variability in the estimate considering user effect
user_avg %>% ggplot(aes(b_u)) +
  geom_histogram(bins = 10, color = "black", fill= "light blue")
```



Re-construct predictor and see RMSE improves:

```
## [1] 0.8552384
```

Method	RMSE
Target RMSE	0.8649000
Simple Average Model	1.0599043
Movie Effect Model	0.9437429
Movie+User Effect Model	0.8552384

Above table shows the estimated effect of user (b_u) building on the movie effects model above. Whilst b_u showed less variability than was observed with b_i, it was evident that adjusting for user effects enhanced the accuracy of the algorithm. Indeed, adjusting for user effects resulted in an RMSE of 0.85524. Thus, adjusting for both movie and user effects improved the RMSE versus the previous two models, demonstrating the strong bias introduced by each of these variables on ratings.

3.2.4 Movie, User & Genre effect model

Movie ratings were also dependent on genre, with some genres achieving higher average ratings than others. This effect was observed even when movies were allocated to multiple genres, as in the original dataset. Therefore, the rating for each movie and user was further refined by adjusting for genre effect, b_g, and the least squares estimate of the genre effect, $\hat{b}_g$ calculated using the formula shown below.

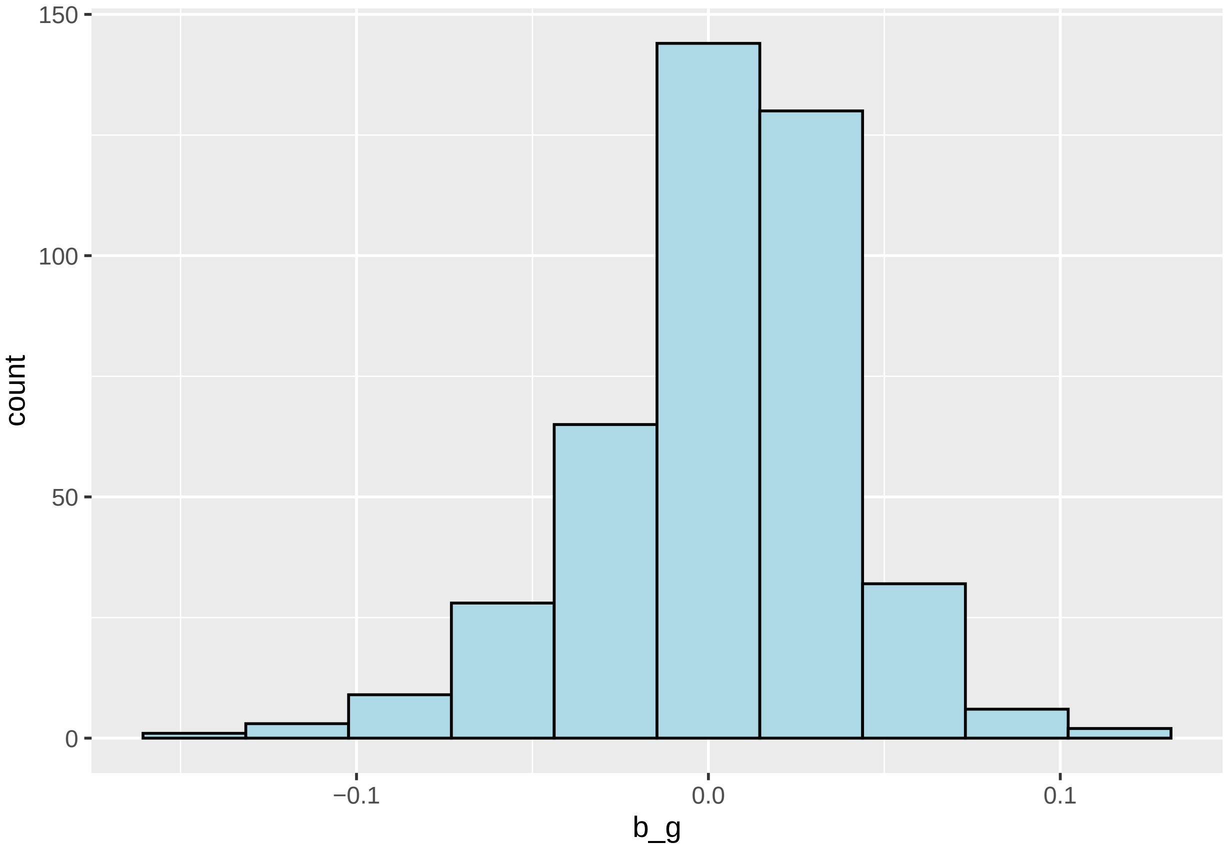
$$Y_{u,i} = \mu + b_i + b_u + b_g + \epsilon_{u,i}$$

$$\hat{b}_g = mean\left(\hat{y}_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u\right)$$

```
mu_genre_avg <- mean(train_set$rating)

# Estimate genre effect (b_g)
genre_avg <- train_set %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by="userId") %>%
  group_by(genres) %>%
  filter(n()>=1000) %>%     #where a particular Genre has at least 1000 count in rating
  summarise(b_g = mean(rating - mu_genre_avg - b_i - b_u, na.rm = TRUE),
            .groups = "drop")    #ensure proper grouping behavior
genre_avg
```

```
## # A tibble: 420 x 2
##    genres                                               b_g
##    <chr>                                              <dbl>
##  1 Action                                          -0.0546
##  2 Action|Adventure                                -0.0106
##  3 Action|Adventure|Animation|Children|Comedy       0.0129
##  4 Action|Adventure|Animation|Comedy|Drama          0.0410
##  5 Action|Adventure|Animation|Drama|Fantasy         0.0341
##  6 Action|Adventure|Animation|Horror|Sci-Fi        -0.0541
##  7 Action|Adventure|Animation|Sci-Fi               -0.00149
##  8 Action|Adventure|Animation|Sci-Fi|Thriller      -0.0147
##  9 Action|Adventure|Children|Comedy                -0.0216
## 10 Action|Adventure|Children|Comedy|Fantasy|Sci-Fi -0.0301
## # i 410 more rows
```

```
#plot variability in the estimate considering Genre effect
genre_avg %>% ggplot(aes(b_g)) +
              geom_histogram(bins = 10, color = "black", fill= "light blue")
```



Re-construct predictor and see how RMSE changes:

Method	RMSE
Target RMSE	0.8649000
Simple Average Model	1.0599043
Movie Effect Model	0.9437429
Movie+User Effect Model	0.8552384
Movie+User+Genre Effect Model	0.8546828

The output from the model when adjusting for genre, in addition to movie and user bias, was an RMSE of 0.85468. Thus adding genre effects into the model only provided a modest improvement in the accuracy of the algorithm.

3.2.5 Movie, User, Genre & Review Date effect model

The final bias to be adjusted for was the date of review. The exploratory analysis had shown that this had a small impact on the average rating awarded for each movie and user. The most appropriate way to incorporate this into the model would be to apply a smooth function to the day of release for each rating by movie and user. Rounding the date of review to the nearest week served to effectively smooth the data. The least squares estimate, taking into account the date of review effect, $\hat{b}_r$ was calculated using the formula shown below.

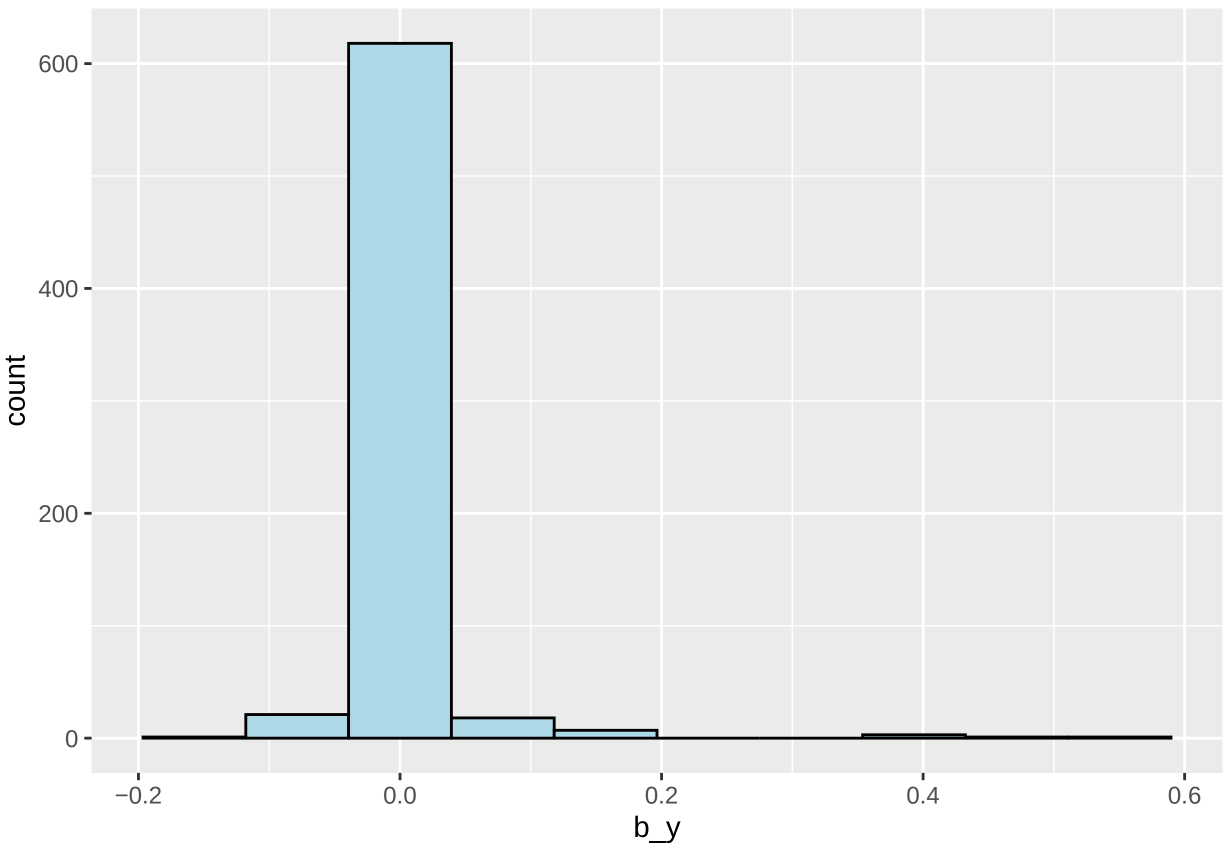
$$Y_{u,i} = \mu + b_i + b_u + b_g + b_y + b_r + \epsilon_{u,i}$$

$$\hat{b}_r = mean\left(\hat{y}_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u - \hat{b}_g - \hat{b}_y\right)$$

```
mu_rev_date_avg <- mean(train_set$rating)

# Estimate Review Year effect (b_y)
rev_date_avg <- train_set %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  left_join(genre_avg, by="genres") %>%
  group_by(review_date) %>%
  summarise(b_y = mean(rating - mu_rev_date_avg - b_i - b_u - b_g, na.rm = TRUE),
            .groups = "drop")     # Handle NA values during summarization
rev_date_avg
```

```
## # A tibble: 671 x 2
##    review_date              b_y
##    <dttm>                 <dbl>
##  1 1995-01-08 00:00:00 NaN
##  2 1996-01-28 00:00:00   0.187
##  3 1996-02-04 00:00:00   0.126
##  4 1996-02-11 00:00:00   0.395
##  5 1996-02-18 00:00:00   0.454
##  6 1996-02-25 00:00:00   0.380
##  7 1996-03-03 00:00:00   0.0125
##  8 1996-03-10 00:00:00   0.355
##  9 1996-03-17 00:00:00   0.128
## 10 1996-03-24 00:00:00   0.185
## # i 661 more rows
```

```
#plot variability in the estimate considering date of review effect
rev_date_avg %>% ggplot(aes(b_y)) +
  geom_histogram(bins = 10, color = "black", fill= "light blue")
```



Re-construct predictor and see how RMSE changes:

Method	RMSE
Target RMSE	0.8649000
Simple Average Model	1.0599043
Movie Effect Model	0.9437429
Movie+User Effect Model	0.8552384
Movie+User+Genre Effect Model	0.8546828
Movie+User+Genre+Review Date Effect Model	0.8545693

Adding the effect of review date into the algorithm delivered an RMSE of 0.85457 and well within the target RMSE established earlier to meet the project objective.

3.2.6 Effect of Regularization

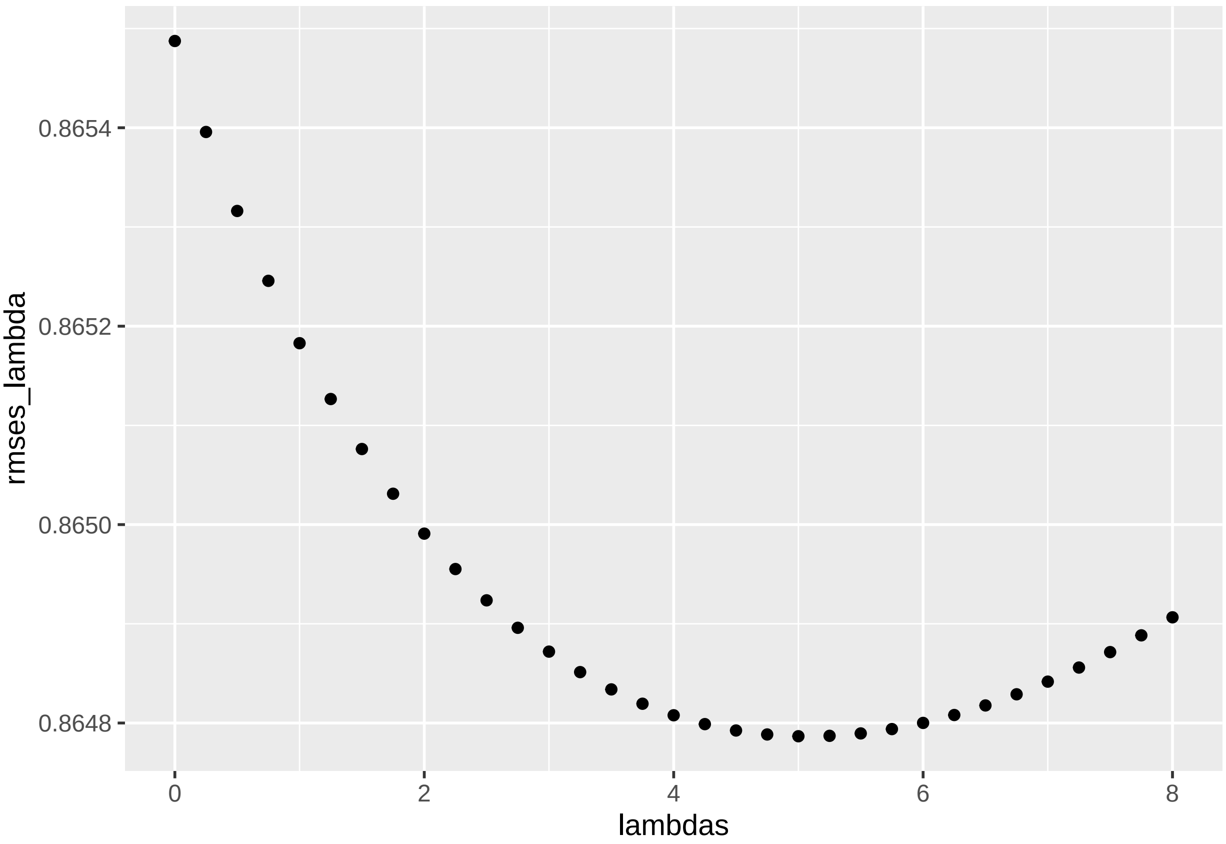
Finally, the exploratory analysis showed that not only is the average rating affected by the movie, user, genre and date of review, but that the number of ratings also varies. Thus, for example, some movies and genres of movie received fewer ratings than others while some users provided fewer ratings than others. Similarly, the number of ratings varied by date of review. In each of these cases, the consequence of this variation is that the estimates of the effect (b) will have been subject to greater uncertainty when based on a smaller number of ratings.

Regularization is an effective method for penalising large effect estimates that are based on small sample sizes. The penalty term, λ, is a tuning parameter chosen using cross-validation within the edx dataset. Thus, the movie, user, genre, review date effect can be regularized to penalise these large effects.

```r
# Here, we use regularization to take into account the number of ratings per movie
# to diminish the b_i effect of movies with a small number of ratings

# Create a grid for the tuning parameter lambda

lambdas <- seq(0, 8, 0.25)

# For each lambda,find b_i & b_u, followed by rating prediction & testing.

rmses_lambda <- sapply(lambdas, function(l){

mu_lambda <- mean(train_set$rating)

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

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

b_g <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarise(b_g = sum(rating - b_i - b_u - mu_lambda, na.rm = TRUE)/(n()+l))

b_y <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    group_by(review_date) %>%
    summarise(b_y = sum(rating - b_i - b_u - b_g - mu_lambda, na.rm = TRUE)/(n()+l))

predicted_ratings_lambda <- test_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    left_join(b_y, by="review_date") %>%
    mutate(pred = mu_lambda + b_i + b_u + b_g + b_y) %>%
    pull(pred)
  return(RMSE(predicted_ratings_lambda, test_set$rating))
})

# This execution for optimal lambdas might take ~5 min. So pls be patient! :-)
```

We plot RMSE vs lambdas to select the optimal lambda

```
# Plot rmses vs lambdas to select the optimal lambda
qplot(lambdas, rmses_lambda)
```



For the full model, the optimal lambda is:

```
# Assign optimal tuning parameter (lambda)
lambda <- lambdas[which.min(rmses_lambda)]
lambda
```

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

For the full model, the optimal lambda is: 5.0

```
# Minimum RMSE achieved using lambdas
regularised_rmse <- min(rmses_lambda)
regularised_rmse
```

```
## [1] 0.8647867
```

```r
# Test and save results
rmse_computed <- bind_rows(rmse_computed,
                tibble(Method="Regularized model", RMSE = regularised_rmse))

rmse_computed %>% knitr::kable()
```

Method	RMSE
Target RMSE	0.8649000
Simple Average Model	1.0599043
Movie Effect Model	0.9437429
Movie+User Effect Model	0.8552384
Movie+User+Genre Effect Model	0.8546828
Movie+User+Genre+Review Date Effect Model	0.8545693
Regularized model	0.8647867

We observe that the RMSE value has increased, although slightly, after applying regularization principles. This can potentially happen due to the various reasons:

Over-Regularization: Regularization is used to penalize overly complex models to avoid over fitting. However, if the regularization parameter λ is too large, it can oversimplify the model, leading to under fitting. Since RMSE increased, although slightly, it suggests that the chosen λ might have been too high.

Model Alignment: The baseline model, without regularization might already be well-tuned to the data. Adding regularization, even when properly configured, might not improve performance significantly still the change is not significant enough to warrant removing regularization principles entirely. Regularization remains valuable for enhancing model robustness, particularly in safeguarding against potential variations or shifts in the dataset over time.

3.3 Final Results

```r
###########################################################
##### Apply final model to predict ratings in final_holdout_test set #####
###########################################################

# Mutate final_holdout_test dataset according to the changes made to edx
final_holdout_test <- final_holdout_test %>%
  mutate(review_date = round_date(as_datetime(timestamp), unit = "week"))

final_predicted_ratings <- final_holdout_test %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(genre_avg, by="genres") %>%
  left_join(rev_date_avg, by="review_date") %>%
  mutate(pred = mu_simple_avg + b_i + b_u + b_g + b_y ) %>%
  pull(pred)
```

RMSE of the final_holdout_test set:

Method	RMSE
Target RMSE	0.8649000
Simple Average Model	1.0599043
Movie Effect Model	0.9437429
Movie+User Effect Model	0.8552384
Movie+User+Genre Effect Model	0.8546828
Movie+User+Genre+Review Date Effect Model	0.8545693
Regularized model	0.8647867
Final Results from Target Data Set	0.8553213

4 Conclusion

The primary objective of this project was to create a recommendation system using the MovieLens 10M dataset that predicted movie ratings with a Root Mean Square Error (RMSE) below 0.86490. By systematically adjusting for biases introduced by factors such as movies, users, genres, release years, and review dates—and applying regularization to control the variability of these effects—the model successfully met the objective, achieving an RMSE of r round(regularised_rmse,5). This performance was further validated using an independent test on the reserved validation dataset, which produced an RMSE of r round(target_set_rmse,5).

While the developed algorithm met the project goal, the remaining error suggests there is room for improvement. Not all residual error may be independent, indicating the potential to enhance accuracy further. A promising next step is the application of matrix factorization, a powerful technique for collaborative filtering that identifies latent patterns among users and movies. By leveraging these patterns, matrix factorization can effectively reduce residual error in predictions (Irizarry 2020; Koren, Bell, and Volinsky 2009).

The scope of techniques implemented in this project was constrained by the computational limitations of processing large datasets on a personal computer. Matrix factorization, however, offers significant advantages in terms of scalability, memory efficiency, and compatibility with such environments. Consequently, future work on this recommendation system would focus on integrating matrix factorization to further enhance its performance and robustness.

References

Harper, F. Maxwell, and Joseph A. Konstan. 2015. "The MovieLens Datasets: History and Context." *ACM Trans. Interact. Intell. Syst.* 5 (4). https://doi.org/10.1145/2827872.

Irizarry, Rafael A. 2020. *Introduction to Data Science: Data Analysis and Prediction Algorithms with r.* CRC Press.

Koren, Y. 2009. "The BellKor Solution to the Netflix Grand Prize." In *Netflix Prize Documentation*. Vol. 81.

Koren, Y., R. Bell, and C. Volinsky. 2009. "Matrix Factorization Techniques for Recommender Systems." *Computer* 42 (8): 30–37.

Lohr, S. 2009. *The New York Times*. The New York Times. https://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest.