HarvardX Capstone Project: MovieLens Recommender System

Mujahid Ali

2023-12-21

Contents

1 Introduction

This project is an essential part of the HarvardX Data Science Capstone Course, and it draws inspiration from challenges similar to the well-known Netflix challenge. The main objective is to develop a user-friendly movie recommendation system. This system will utilize the expansive Movielens dataset, which contains over 10 million movie ratings, provided by the Department of Computer Science and Engineering at the University of Minnesota.

The Movielens dataset is rich and diverse, encompassing a broad range of movies. Each movie in the dataset is detailed with its title, genre, and associated user ratings, which include the time of rating. The project focuses on creating a reliable predictive model that can accurately estimate unprovided movie ratings. The success and accuracy of this predictive model will be evaluated using the Root Mean Square Error ($RMSE$) metric, a standard measure in machine learning for gauging the performance of prediction models.

Through this project, we aim to not only develop an effective recommendation system but also explore and understand patterns and trends in movie ratings. This could offer valuable insights into user preferences and behaviors, enhancing the overall movie-viewing experience. The application of various data science techniques and machine learning models will play a crucial role in achieving these objectives.

2 Methodology

In this section, we detail the methodology employed in this project, encompassing three critical stages: data preparation, model development, and model evaluation.

The data preparation phase involves meticulously processing the Movielens dataset. This includes cleaning the data, handling any missing or inconsistent entries, and performing exploratory data analysis to gain insights. We also engineer relevant features that could enhance the predictive power of our models.

Following data preparation, we embark on constructing various predictive models. Starting with basic models, we gradually build more complex and refined models, each aiming to accurately predict movie ratings. These models range from simple linear regression to more advanced machine learning techniques, adapting and improving based on the insights gained from the data.

The final phase focuses on evaluating the performance of these models. We use the Root Mean Square Error ($RMSE$) as our primary metric to assess each model's accuracy in predicting movie ratings. This evaluation helps us determine the most effective model and fine-tune it for optimal performance.

This methodical approach ensures a comprehensive understanding of the dataset, the application of appropriate modeling techniques, and the selection of the best model based on empirical evidence.

2.1 Overview

The development of the model is done in three main steps.

1. Data Load, Analyis and Preparation

We begin by loading the Movielens datasets, as provided in the course materials. This initiates a series of straightforward yet crucial analyses to understand the dataset's structure and characteristics. Key in this phase is the computation of additional variables that serve as predictors, enhancing the potential of more advanced modeling techniques. Visual representations are also used to present these variables clearly. The dataset is then strategically divided: one part for model building (further split into training and testing sets) and another part for validation. A smaller subset is also created to ensure efficiency in later stages of model development.

2. Model Development

This phase involves a progressive approach to building prediction models: - We start with a basic linear model using movie and user data as predictors. - This model is then refined through regularization to improve its performance. - Next, we develop a more complex regularized linear model, introducing two new predictors: the count of ratings per movie and the time elapsed since the movie's first rating. - We then create a generalized linear model to address the residuals of the previous model. - Finally, we assemble an ensemble of various models to further enhance prediction accuracy, focusing on the residuals from the last model.

3. Model Validation

The validation phase is critical in testing the effectiveness of our models. For this, we use a separate subset of data reserved exclusively for validation. This approach ensures that our models are tested under realistic conditions, providing us with reliable measures of their performance.

2.2 Data Load, Analysis and Preparation

We start by loading essential libraries for data manipulation and analysis:

- *tidyverse*: A suite of packages for data manipulation and visualization.
- *caret*: Offers functions for creating predictive models.
- *doParallel*: Enables parallel computing.
- *timeDate*: Useful for handling date-time data.
- *stringi*: Provides fast and flexible string operations.
- *gam*: Used for generalized additive models.
- *scales*: Helps in formatting graph axes.
- *ggthemes*: Enhances ggplot graphs with additional themes.

2.2.1 Data Loading

Next, we load the Movielens dataset, provided for this project.

```
# MovieLens dataset
options(timeout = 120)
dataset_url <- "https://files.grouplens.org/datasets/movielens/ml-10m.zip"
dataset_zip <- "ml-10M100K.zip"

# Download and unzip if not already done
if (!file.exists(dataset_zip)) {
  download.file(dataset_url, dataset_zip)
}
if (!file.exists("ml-10M100K/ratings.dat")) {
  unzip(dataset_zip, "ml-10M100K/ratings.dat")
}
if (!file.exists("ml-10M100K/movies.dat")) {
  unzip(dataset_zip, "ml-10M100K/movies.dat")
}

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")
```

2.2.2 Data Analysis

Now, we dive into the dataset for a deeper understanding.

```
# Exploring the dataset structure
str(movielens)
```

```
## 'data.frame':    10000054 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 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
```

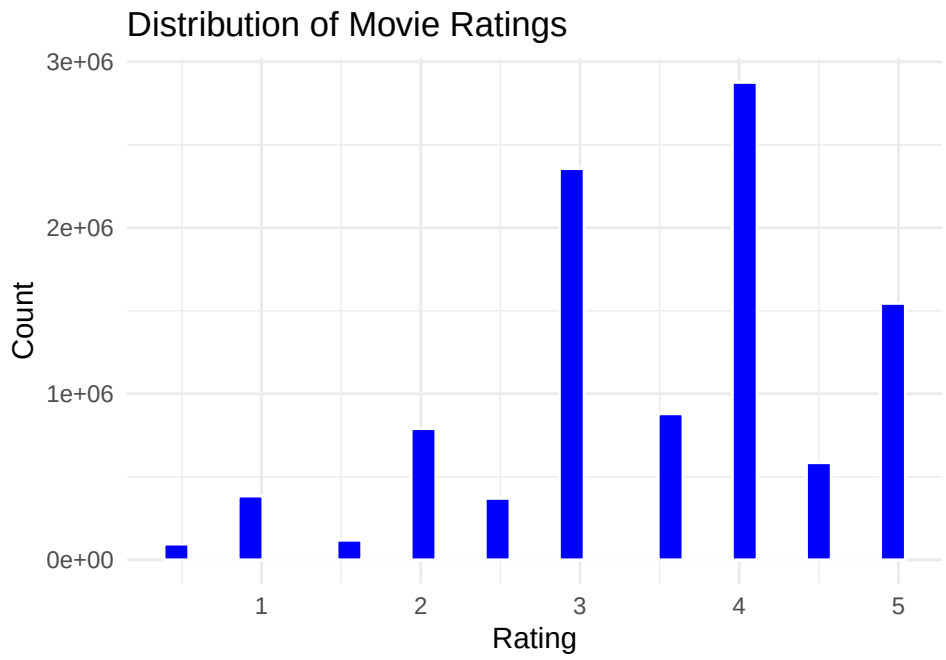
```
# Counting unique movies and users
num_movies <- n_distinct(movielens$movieId)
num_users <- n_distinct(movielens$userId)
cat("Number of movies:", num_movies, "\nNumber of users:", num_users, "\n")
```

```
## Number of movies: 10677
## Number of users: 69878
```

```r
# Calculating the average rating
average_rating <- mean(movielens$rating)
cat("Average movie rating:", average_rating, "\n")
```

```
## Average movie rating: 3.5124
```

```r
# Visualizing the distribution of ratings
ggplot(movielens, aes(x = rating)) +
  geom_histogram(bins = 30, fill = "blue", color = "white") +
  theme_minimal() +
  labs(title = "Distribution of Movie Ratings", x = "Rating", y = "Count")
```




```r
# Analysis of Top 5 Most-Rated Movies
```

We examine the most popular movies in the Movielens dataset, identified by the highest number of ratings received. This analysis helps us understand which movies attracted the most viewer attention and how their average ratings compare to the general trend.

```
## # A tibble: 5 x 4
##   movieId count   avg title
##     <int> <int> <dbl> <chr>
## 1     296 34864  4.16 Pulp Fiction (1994)
## 2     356 34457  4.01 Forrest Gump (1994)
## 3     593 33668  4.20 Silence of the Lambs, The (1991)
## 4     480 32631  3.66 Jurassic Park (1993)
## 5     318 31126  4.46 Shawshank Redemption, The (1994)
```

Contrasting with the most popular movies, an examination of the top 5 movies with the highest ratings reveals an interesting pattern. These films, while achieving high ratings, tend to be less recognized and have garnered only a limited number of ratings in comparison.

This observation suggests that high-quality cinema may not always align with widespread popularity. Such movies, although they may not have a vast audience or the same level of visibility as blockbuster hits, often resonate strongly with a smaller, more niche audience, resulting in higher average ratings. This aspect of movie ratings highlights the diversity of audience preferences and the varied factors that contribute to a movie's appeal and rating.

```
## # A tibble: 5 x 4
##   movieId count   avg title
##     <int> <int> <dbl> <chr>
## 1   33264     2     5 Satan's Tango (Sátántangó) (1994)
## 2   42783     1     5 Shadows of Forgotten Ancestors (1964)
## 3   51209     1     5 Fighting Elegy (Kenka erejii) (1966)
## 4   53355     1     5 Sun Alley (Sonnenallee) (1999)
## 5   64275     1     5 Blue Light, The (Das Blaue Licht) (1932)
```

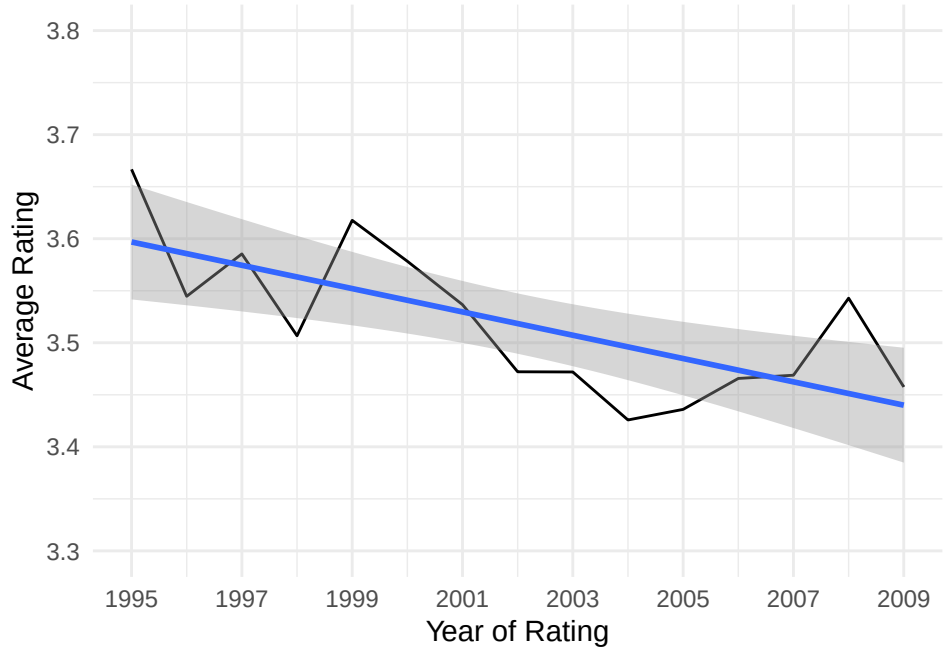
2.2.3 Enhanced Data Preparation for Advanced Modeling

As part of enhancing our predictive models, we compute several additional variables and incorporate them into the dataset. These enhancements, executed before dividing the data into modeling and validation sets, are crucial for the effectiveness of our advanced models. They provide additional predictive power and depth to our analysis.

```
# Analyzing Rating Trends Over Time
```

To gain insights into temporal trends, we add the date and year of each rating. A subsequent analysis of average ratings across years reveals an intriguing trend: a gradual decline in ratings over time. This trend analysis offers a temporal dimension to our understanding of movie ratings, suggesting changes in user preferences or rating behaviors over the years.

```
# Adding rating date and year to the dataset
movielens <- movielens %>%
  mutate(rating_date = as.Date(as.POSIXct(timestamp, origin = "1970-01-01")),
         rating_year = year(rating_date))

# Visualization: Average Rating by Year
# Plotting the trend of average ratings across different years
movielens %>%
  group_by(rating_year) %>%
  summarize(AverageRating = mean(rating)) %>%
  ggplot(aes(x = rating_year, y = AverageRating)) +
  geom_line() +
  geom_smooth(formula = y ~ x, method = "lm", na.rm = TRUE) +
  ylim(3.3, 3.8) +
  labs(y = "Average Rating", x = "Year of Rating") +
  scale_x_continuous(breaks = seq(1995, 2010, 2)) +
  theme_minimal()
```



Extracting and Analyzing Movie Release Years

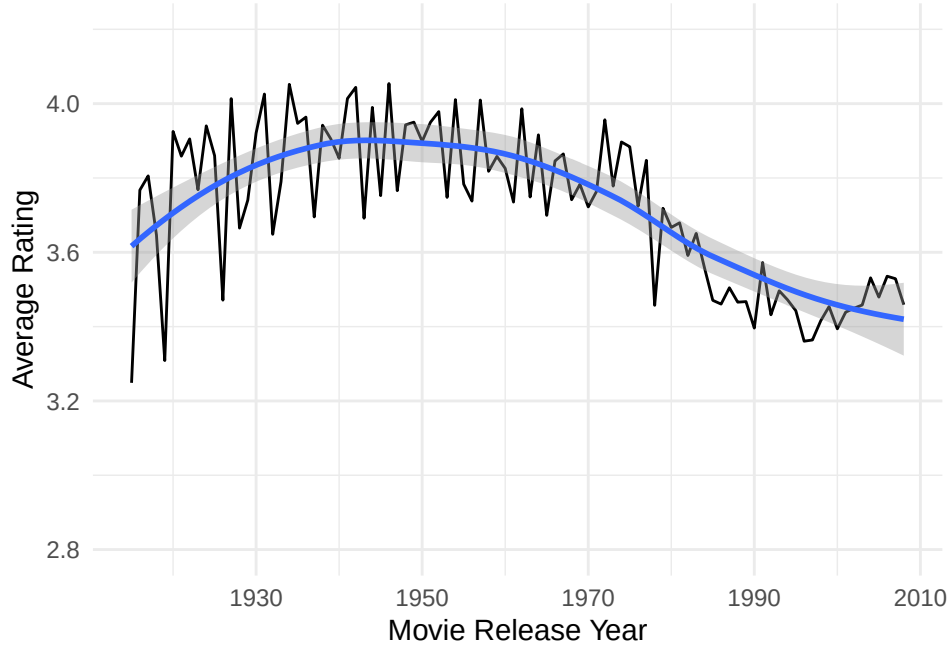
The movie release year, encoded within the movie title in the format (yyyy), plays a crucial role in our analysis. We utilize regular expressions to extract this information. A validation step ensures all movie titles correctly follow this format.

Subsequent analysis focuses on understanding the relationship between a movie's release year and its average rating. Intriguingly, older movies often receive higher ratings, except for some early 20th-century films. We also calculate the age of each movie at the time it was rated to explore this aspect further.

```
# Regular expression pattern for extracting release year
pattern <- "\\(\\d{4}\\)$"

# Validate the pattern in movie titles
if (0 < sum(!str_detect(movielens$title, pattern))) {
  stop("Error extracting years from movie titles")
}

# Extract release year and compute movie age at the time of rating
movielens <- movielens %>%
  mutate(movie_year = as.integer(substr(str_match(title, pattern), 2, 5)),
         movie_age = rating_year - movie_year)

# Visualization: Impact of Movie Release Year on Average Rating
movielens %>%
  group_by(movie_year) %>%
  summarize(AverageRating = mean(rating)) %>%
  ggplot(aes(x = movie_year, y = AverageRating)) +
  geom_line() +
  geom_smooth(method = "loess", na.rm = TRUE) +
  labs(y = "Average Rating", x = "Movie Release Year") +
  ylim(2.8, 4.2) +
```

```
  theme_minimal()
```

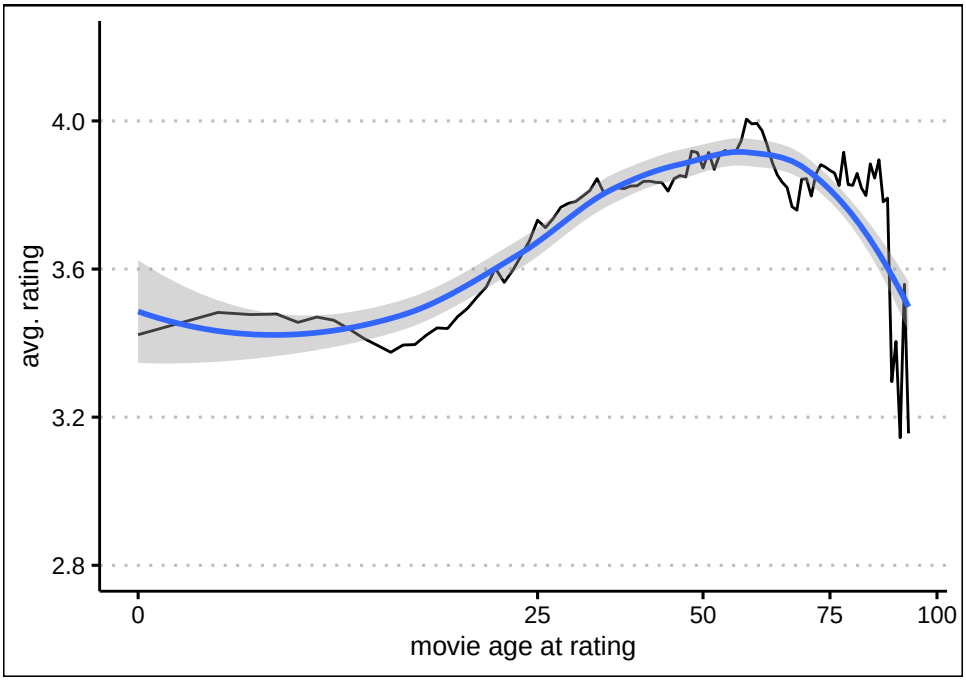
```
## `geom_smooth()` using formula = 'y ~ x'
```




```
# Verify and correct movie ages
# Check for instances where ratings occurred before the movie's release
rating_before_release <- movielens |>
  filter(movie_age < 0) |>
  summarize(n = n_distinct(movieId)) |>
  pull(n)

if (0 <  rating_before_release) {
  warning (paste ("Found", rating_before_release,
    "movies with ratings before release date."))

  # set movie_age to 0 if negative
  movielens <- movielens |>
    group_by(movie_age) |>
    mutate(movie_age = max(0, movie_age)) |>
    ungroup()
}
```

```
## Warning: Found 23 movies with ratings before release date.
```

```
# Visualization: Average Rating by Movie Age at Rating Date
# Exploring how the age of a movie at the time of rating influences its average rating
```



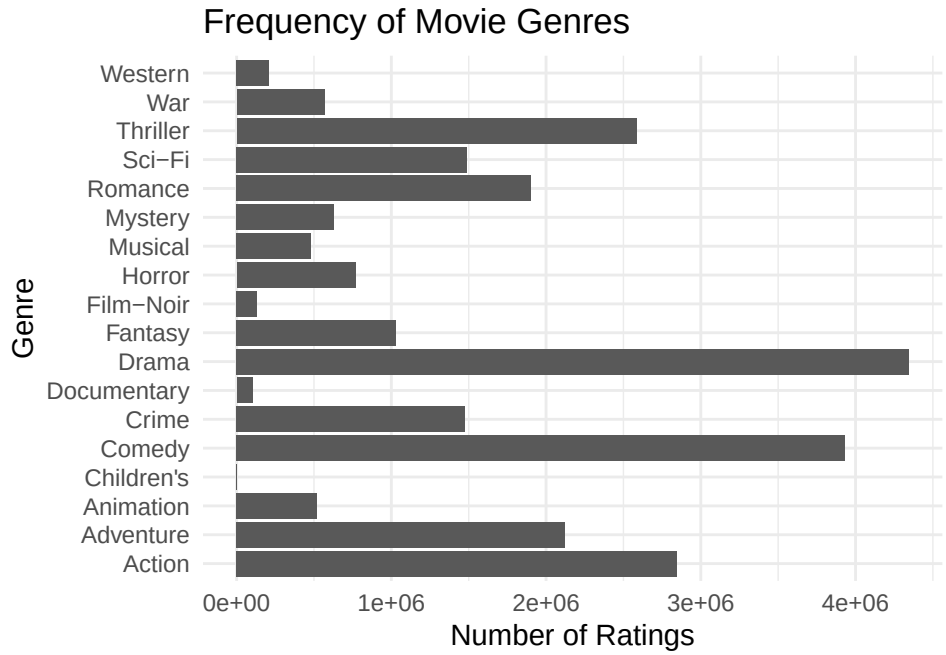
Genre Analysis and Preparation for Modeling

An exploration of movie genres forms our next analytical step. In the Movielens dataset, each movie might be tagged with multiple genres, formatted as "genre1|genre2|...". The dataset encompasses a diverse range of genres, each contributing to the movie's identity.

As per Movielens documentation, genres include Action, Adventure, Animation, and others. We focus on the top five genres - Drama, Comedy, Action, Thriller, and Adventure - for their prevalence in the dataset. To aid in our predictive modeling, we create binary (dichotomous) variables for these genres, where a value of 1 signifies a movie's association with a specific genre.

```
# Counting occurrences of each genre
all_genres <- c("Action", "Adventure", "Animation", "Children's", "Comedy", "Crime",
                "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror", "Musical",
                "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western")

genre_counts <- sapply(all_genres, function(genre) {
  sum(stri_detect_fixed(movielens$genres, genre))
})


# Count genres in Movielens genres, format is "genre1|genre2|..."
all_genres <- c("Action", "Adventure", "Animation", "Children's", "Comedy", "Crime",
                "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror", "Musical",
                "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western")
# Creating a data frame for genre counts
genre_counts_df <- data.frame(Genre = names(genre_counts), Count = genre_counts) %>%
  arrange(desc(Count))

# Visualizing the frequency of each genre
genre_counts_df %>%
  ggplot(aes(x = Genre, y = Count)) +
```

```
  geom_bar(stat = "identity") +
  coord_flip() +
  labs(y = "Number of Ratings", title = "Frequency of Movie Genres") +
  theme_minimal()
```

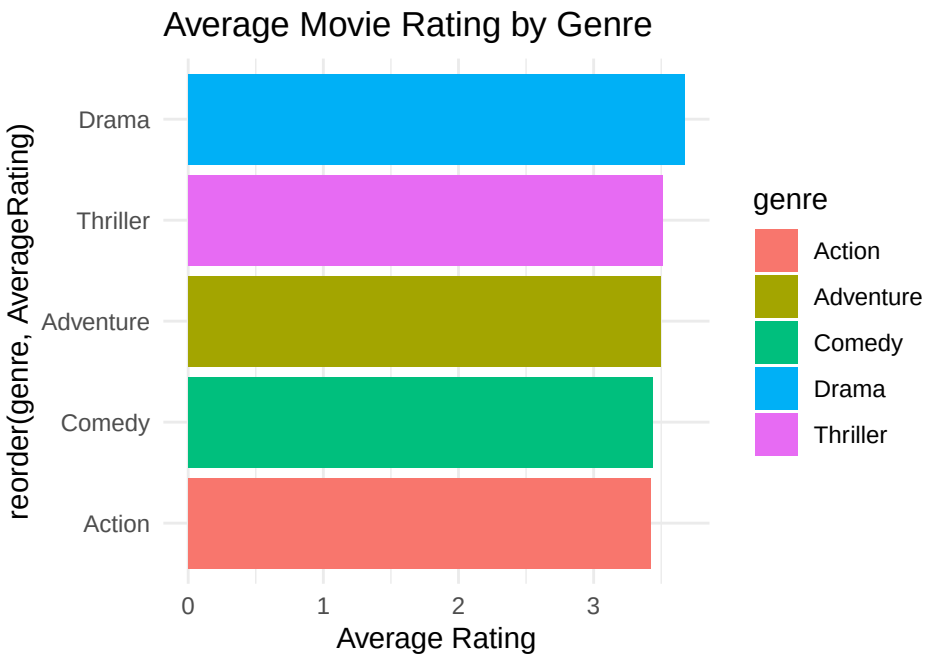



```
# Identifying the top 5 genres
top5_genres <- head(genre_counts_df$Genre, 5)

# Creating binary variables for the top 5 genres
for (genre in top5_genres) {
  column_name <- paste("genre", genre, sep = "_")
  movielens[[column_name]] <- as.integer(stri_detect_fixed(movielens$genres, genre))
}

# Visualizing Average Ratings by Top Genres
```

Next, we assess how these top genres correlate with average ratings. This visualization offers insights into genre popularity and audience preferences.

```
# Preparing data for visualization
genre_rating_data <- movielens %>%
  select(one_of(c("userId", "movieId", "rating", paste("genre_", top5_genres, sep = "")))) %>%
  pivot_longer(cols = starts_with("genre_"), names_to = "genre", names_prefix = "genre_", values_to = "
  filter(is_genre == 1) %>%
  group_by(genre) %>%
  summarize(AverageRating = mean(rating))

# Plotting average rating for each of the top 5 genres
genre_rating_data %>%
  ggplot(aes(x = reorder(genre, AverageRating), y = AverageRating, fill = genre)) +
  geom_col() +
```

```
  coord_flip() +
  labs(y = "Average Rating", title = "Average Movie Rating by Genre") +
  theme_minimal()
```



Integrating Time-Based Variables for Enhanced Modeling

To refine our modeling, we introduce additional time-based variables into the Movielens dataset. These variables capture the temporal aspects of user behavior and movie ratings, providing deeper insights for sophisticated modeling.

We start by calculating the date of each user's first movie rating. From this, we derive the time gap (in days) between each subsequent rating and their first. This metric, termed as "rating experience," reflects how user preferences might evolve over time.

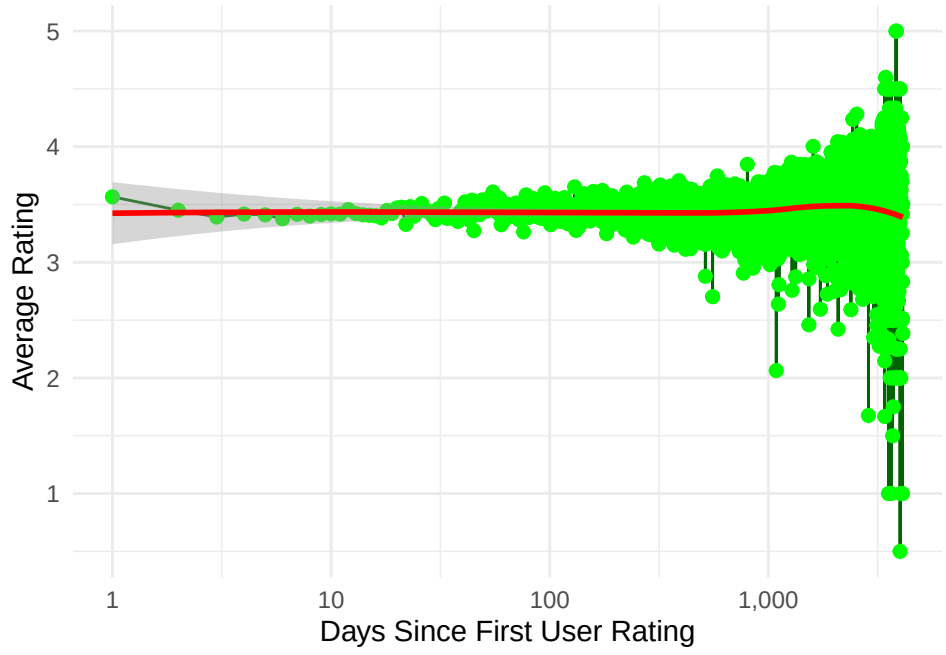
Similarly, we determine the date when each movie received its first rating. The time difference between this date and subsequent ratings is also calculated, offering a perspective on how movie reception changes over time. Additionally, we compute the total number of ratings for each user and movie, as these counts are indicative of user engagement and movie popularity.

Visualizations are created to analyze how these temporal factors influence average ratings, shedding light on trends and patterns in user engagement and movie reception.

```
# Calculating the date of first rating for each user and movie
movielens <- movielens %>%
  group_by(userId) %>%
  mutate(user_first_rating = min(rating_date)) %>%
  ungroup() %>%
  group_by(movieId) %>%
  mutate(movie_first_rating = min(rating_date)) %>%
  ungroup()

# Computing time gaps and rating counts
```

```r
movielens <- movielens %>%
  mutate(days_since_user_first_rating = as.integer(rating_date - user_first_rating) + 1,
         days_since_movie_first_rating = as.integer(rating_date - movie_first_rating) + 1) %>%
  group_by(userId) %>%
  mutate(user_rating_count = n()) %>%
  ungroup() %>%
  group_by(movieId) %>%
  mutate(movie_rating_count = n()) %>%
  ungroup()


# Visualization: Average Rating by User's "Rating Experience"
movielens %>%
  group_by(days_since_user_first_rating) %>%
  summarize(AverageRating = mean(rating)) %>%
  ggplot(aes(x = days_since_user_first_rating, y = AverageRating)) +
  geom_line(color = "darkgreen") +
  geom_point(color = "green", size = 2) +
  geom_smooth(method = "loess", color = "red", na.rm = TRUE) +
  scale_x_log10(name = "Days Since First User Rating", labels = comma) +
  ylab("Average Rating") +
  theme_minimal() +
  theme(legend.position = "none")
```

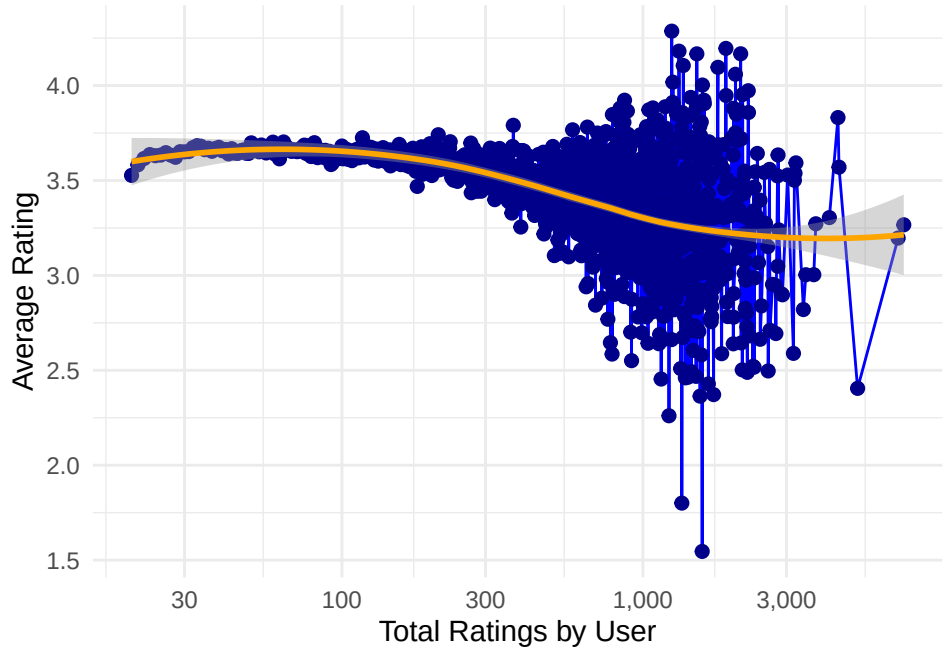
```
## `geom_smooth()` using formula = 'y ~ x'
```




```r
# Visualization: Average Rating by Number of User Ratings
movielens %>%
  group_by(user_rating_count) %>%
  summarize(AverageRating = mean(rating)) %>%
```

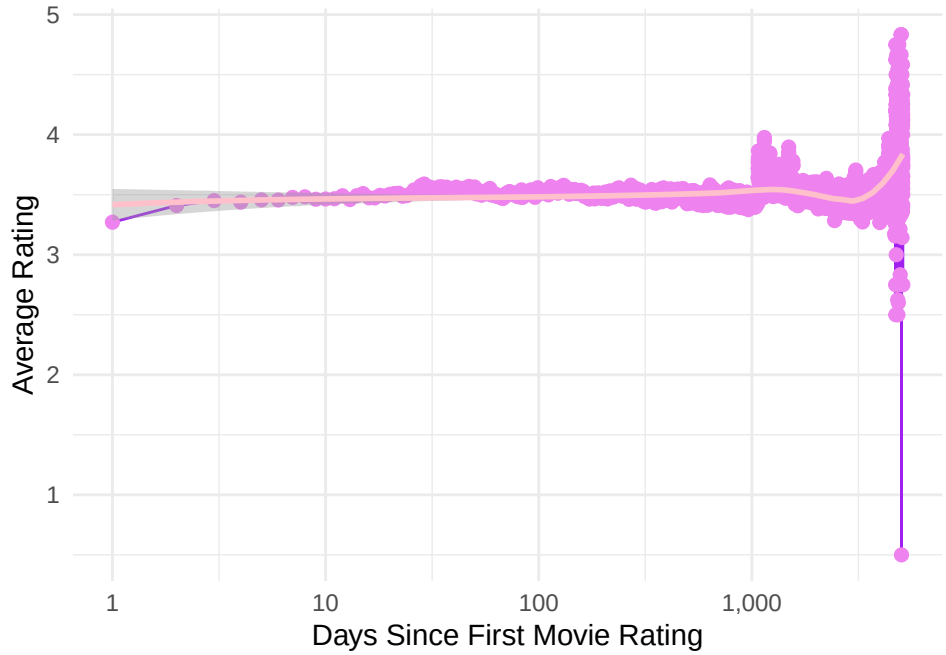
```
  ggplot(aes(x = user_rating_count, y = AverageRating)) +
  geom_line(color = "blue") +
  geom_point(color = "darkblue", size = 2) +
  geom_smooth(method = "loess", color = "orange", na.rm = TRUE) +
  scale_x_log10(name = "Total Ratings by User", labels = comma) +
  ylab("Average Rating") +
  theme_minimal() +
  theme(legend.position = "none")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



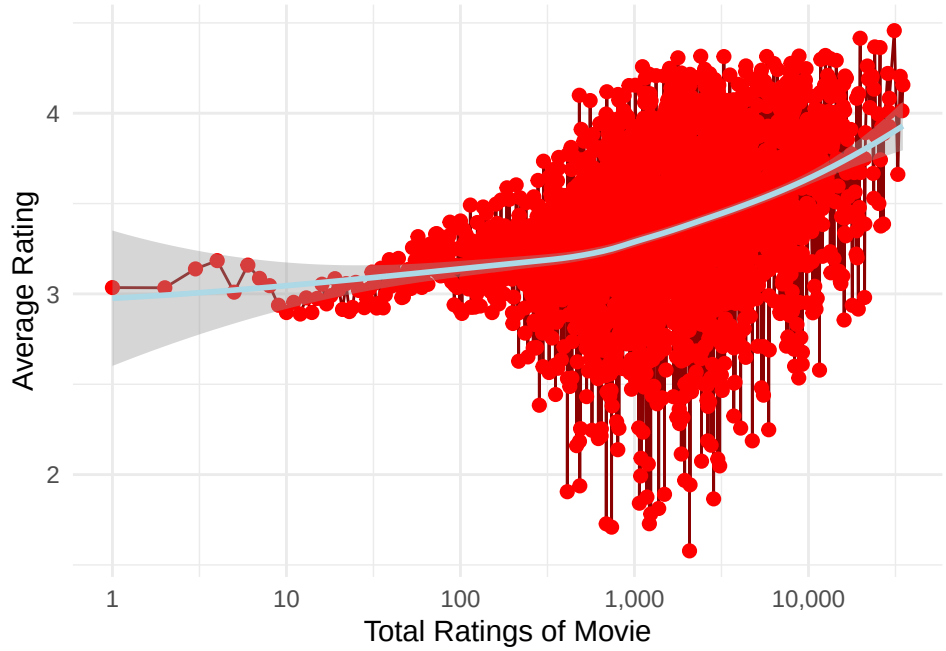

```
# Visualization: Average Rating by Movie's "Rating Duration"
movielens %>%
  group_by(days_since_movie_first_rating) %>%
  summarize(AverageRating = mean(rating)) %>%
  ggplot(aes(x = days_since_movie_first_rating, y = AverageRating)) +
  geom_line(color = "purple") +
  geom_point(color = "violet", size = 2) +
  geom_smooth(method = "loess", color = "pink", na.rm = TRUE) +
  scale_x_log10(name = "Days Since First Movie Rating", labels = comma) +
  ylab("Average Rating") +
  theme_minimal() +
  theme(legend.position = "none")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




```
# Visualizing Average Rating by Number of Movie Ratings
movielens %>%
  group_by(movie_rating_count) %>%
  summarize(AverageRating = mean(rating)) %>%
  ggplot(aes(x = movie_rating_count, y = AverageRating)) +
  geom_line(color = "darkred") +
  geom_point(color = "red", size = 2) +
  geom_smooth(method = "loess", color = "lightblue", na.rm = TRUE) +
  scale_x_log10(name = "Total Ratings of Movie", labels = comma) +
  ylab("Average Rating") +
  theme_minimal() +
  theme(legend.position = "none")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




```
# Cleaning up the dataset by removing unnecessary columns
movielens <- movielens %>%
  select(-title, -genres, -timestamp, -rating_date, -user_first_rating, -movie_first_rating)
```

2.2.4 Splitting Data into Model and Validation Sets

We now partition our dataset into two segments: one for model development and another for validation. This step is crucial to ensure that our model is tested on unseen data, thus providing a realistic evaluation of its performance.

```
#### Final hold-out test set will be 10% of MovieLens data
# Setting seed for reproducibility
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler used
```

```
# set.seed(1) # if using R 3.5 or earlier

# Splitting data - 10% for final hold-out test set
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Ensuring overlap of userId and movieId in test set and edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Reintegrating excluded data back into edx
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, rating_year, movie_year, movie_age, genre_Drama,
## genre_Adventure, days_since_user_first_rating, days_since_movie_first_rating, user_rating_count, movi
```

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

# Clean up workspace
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

```
## Warning in rm(dl, ratings, movies, test_index, temp, movielens, removed): object 'dl' not found
```

Creating Training and Testing Datasets

Further dividing our model dataset, edx, we create distinct training and testing datasets. The training set will contain 90% of the data, while the testing set will hold the remaining 10%. This split ensures a thorough evaluation of our models during development.

```
# Further split edx into training and testing sets
set.seed(2)
model_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
model_data_test <- edx[model_index, ]
model_data_train <- edx[-model_index, ]

# Ensuring consistency in training and testing sets
model_data_test <- model_data_test %>%
  semi_join(model_data_train, by = "movieId") %>%
  semi_join(model_data_train, by = "userId") |>
  semi_join(model_data_train, by = "days_since_movie_first_rating") |>
  semi_join(model_data_train, by = "movie_rating_count")
```

Generating a Smaller Dataset for Efficient Processing

To expedite the regularization process, we generate a smaller subset (5% of edx). This subset is further divided into its own training and testing sets for cross-validation, aiding in efficient model tuning and evaluation.

```
# Creating a smaller dataset from edx
set.seed(3)
small_index <- createDataPartition(y = edx$rating, times = 1, p = 0.05, list = FALSE)
small_data <- edx[small_index, ]

# create train and test set for cross-validation from edx_small
set.seed(4)
test_index <- createDataPartition(y = small_data$rating, times = 1, p = 0.2, list = FALSE)
small_data_train <- small_data[-test_index, ]
small_data_test <- small_data[test_index, ]
# Make sure that small_data_test contains only movies, users,
# days_since_movie_first_rating and movie_rating_count with an entry
# in small_data_train
small_data_test <- small_data_test |>
  semi_join(small_data_train, by = "movieId") |>
  semi_join(small_data_train, by = "userId") |>
  semi_join(small_data_train, by = "days_since_movie_first_rating") |>
  semi_join(small_data_train, by = "movie_rating_count")
```

2.3 Developing Predictive Models Step-by-Step

The journey of model development in our project starts with a basic linear model, evolving gradually to more complex and sophisticated models. Each model builds upon the previous one, using the edx dataset. We begin with a simple linear model, then move to a regularized version, and progressively integrate more advanced techniques including General Linear Models (GLM), Random Forests (RF), and Local Regression (loess).

```
# Enabling parallel processing for efficiency
registerDoParallel(cores=detectCores()/2)

# Setting precision for Root Mean Square Error (RMSE) calculations
options(digits = 5)

# Defining the RMSE function for model evaluation
RMSE <- function(true_ratings, predicted_ratings) {
  sqrt(mean((true_ratings - predicted_ratings) ^ 2))
}
```

2.3.1 Building a Simple Linear Model

Our first model, a simple linear model, predicts ratings based on user and movie biases. The formula for the model is:

$$\hat{y} = \hat{\mu} + b_{\text{movie}} + b_{\text{user}} + \text{error}$$

We compute biases for movies and users and then predict ratings by combining these biases.

The model construction adheres to the methodology outlined in the course. Initially, the movie bias is calculated as:

$$b_{\text{movie}} = \frac{1}{n}\sum_i (rating_i - \hat{\mu}) = \sum_i \frac{rating_i - \hat{\mu}}{n}$$

Subsequently, the user bias is computed:

$$b_{\text{user}} = \frac{1}{n}\sum_i (rating_i - \hat{\mu} - b_{\text{movie}}) = \sum_i \frac{rating_i - \hat{\mu} - b_{\text{movie}}}{n}$$

The resulting variables b_{movie} and b_{user} are stored in datasets *fit_movies* and *fit_users*. The prediction is then calculated after joining *fit_movies* and *fit_users* to the test dataset using the following formula:

```
y_hat = mu_hat + b_movie + b_user + error
```

The resulting Root Mean Square Error (*RMSE*) is stored in *evaluation*.

```
# Calculate y_hat
mu_hat <- mean(model_data_train$rating)

# Movie and user effects
model_data_train <- model_data_train |>
  group_by(movieId) |>
```

```
  mutate(b_movie = mean(rating - mu_hat)) |>
  ungroup() |>
  group_by(userId) |>
  mutate(b_user = mean(rating - mu_hat - b_movie)) |>
  ungroup()

# Test model, save b_movie and b_user
fit_movies <- model_data_train |> group_by(movieId) |>
  summarize(b_movie = first(b_movie))

fit_users <- model_data_train |> group_by(userId) |>
  summarize(b_user = first(b_user))

# Compute prediction
rating_hat_simple <- model_data_test |>
  left_join(fit_movies, by = "movieId")  |>
  left_join(fit_users, by = "userId")  |>
  mutate(rating_hat = mu_hat + b_movie + b_user) |>
  pull(rating_hat)

# Output the RMSE evaluation
evaluation <- data.frame(
  data = "model",
  model = "simple linear: mu + b_movie + b_user",
  RMSE = RMSE(model_data_test$rating, rating_hat_simple)
)
evaluation
```

```
##    data                                 model    RMSE
## 1 model simple linear: mu + b_movie + b_user 0.86638
```

2.3.2 Enhancing the Model with Regularization

Advancing from the initial simple linear model, we introduce regularization to mitigate the influence of outliers and overfitting, particularly for movies with fewer ratings. This regularization technique balances the model's complexity with its predictive power.

Regularization is applied by incorporating a shrinkage parameter λ:

$$b_{\text{movie}} = \frac{\sum_i (rating_i - \hat{\mu})}{n + \lambda}$$

Similarly, the user bias with regularization becomes:

$$b_{\text{user}} = \frac{\sum_i (rating_i - \hat{\mu} - b_{\text{movie}})}{n + \lambda}$$

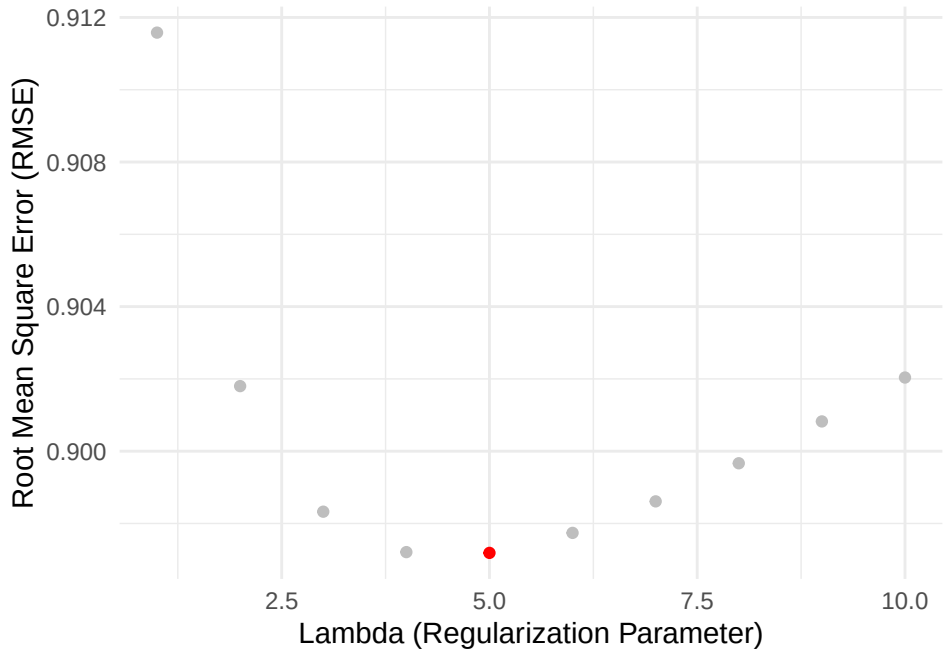
These adjustments temper the biases, especially when data is sparse. The optimal value of λ is identified via cross-validation, ensuring our model remains robust and generalizable.

```
# Define true ratings for cross-validation
rating <- small_data_test |> pull(rating)
```

```r
# Function to train the simple linear model with regularization
train_simple_linear_model <- function(lambda) {
  # Apply regularization to the bias terms within the training subset
  small_data_train <- small_data_train |>
    group_by(movieId) |>
    mutate(b_movie = sum(rating - mu_hat) / (n() + lambda)) |>
    ungroup() |>
    group_by(userId) |>
    mutate(b_user = sum(rating - mu_hat - b_movie) / (n() + lambda)) |>
    ungroup()

  # Prepare for prediction by saving bias terms
  fit_movies <- small_data_train |>
    group_by(movieId) |>
    summarize(b_movie = first(b_movie))

  fit_users <- small_data_train |>
    group_by(userId) |>
    summarize(b_user = first(b_user))

  # Predict ratings and calculate RMSE for cross-validation
  rating_hat <- small_data_test |>
    left_join(fit_movies, by = "movieId") |>
    left_join(fit_users, by = "userId") |>
    mutate(rating_hat = mu_hat + b_movie + b_user) |>
    pull(rating_hat)

  RMSE(rating, rating_hat)
}

# Execute cross-validation across a range of lambda values
lambdas <- seq(1, 10, 1)
rmses <- sapply(lambdas, train_simple_linear_model)
lambda <- lambdas[which.min(rmses)]
print(lambda)
```

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

```r
# Visualize RMSE across different lambda values to find the optimal parameter
data.frame(lambda = lambdas, rmse = rmses) |>
  mutate(is_optimal = lambda == optimal_lambda) |>
  ggplot(aes(x = lambda, y = rmse, color = is_optimal)) +
  geom_point() +
  xlab("Lambda (Regularization Parameter)") +
  ylab("Root Mean Square Error (RMSE)") +
  scale_color_manual(values = c("grey", "red")) +
  guides(color = FALSE) +
  theme_minimal()
```



Upon determining the optimal λ, we retrain our model on the larger dataset, taking into account the regularization effect. The residuals, representing the deviation from the actual ratings, are an essential component for further model enhancements and evaluations.

$$\text{residual} = \text{rating} - \hat{\text{rating}}$$

```
# Regularized bias terms are now computed with the optimal lambda value
model_data_train <- model_data_train |>
  group_by(movieId) |>
  mutate(b_movie = sum(rating - mu_hat)/(n() + lambda)) |>
  ungroup() |>
  group_by(userId) |>
  mutate(b_user = sum(rating - mu_hat - b_movie)/(n() + lambda),
         rating_hat = mu_hat + b_movie + b_user,
         residual = rating - rating_hat) |>
  ungroup()

# Prepare the model for testing by storing the regularized bias terms
fit_movies <- model_data_train |> group_by(movieId) |>
  summarize(b_movie = first(b_movie))

fit_users <- model_data_train |> group_by(userId) |>
  summarize(b_user = first(b_user))

# Predict ratings on the test dataset
rating_hat <- model_data_test |>
  left_join(fit_movies, by = "movieId")  |>
  left_join(fit_users, by = "userId")  |>
  mutate(rating_hat = mu_hat + b_movie + b_user) |>
  pull(rating_hat)

# Update the evaluation data frame with the new RMSE
```

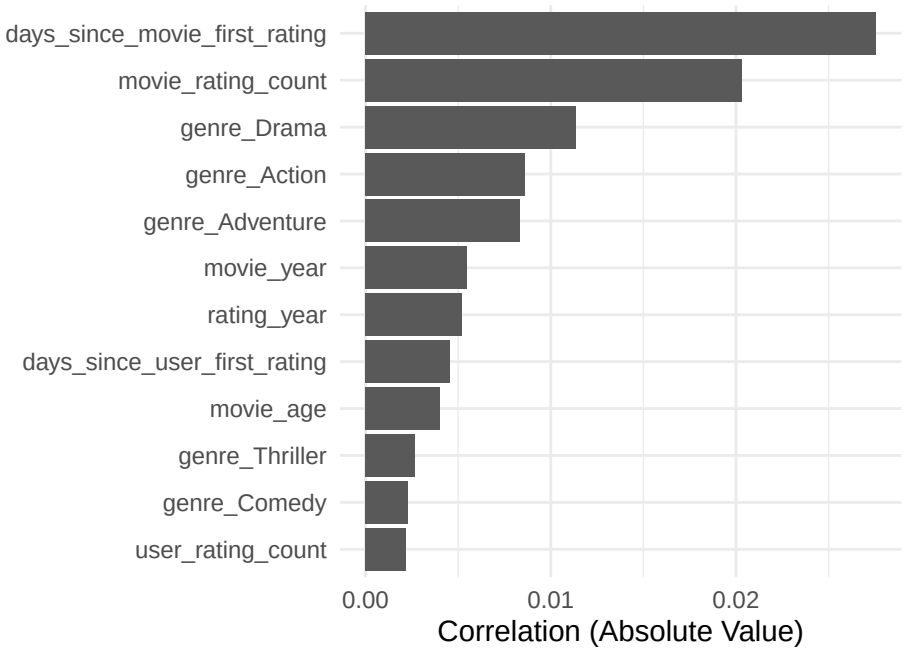
```r
evaluation <- rbind(
  evaluation,
  data.frame(
    data = "model",
    model = "Regularized Linear Model",
    RMSE = RMSE(model_data_test$rating, rating_hat)
  )
)

# Print the updated evaluation for comparison
print(evaluation)
```

```
##    data                                  model    RMSE
## 1 model simple linear: mu + b_movie + b_user 0.86638
## 2 model             Regularized Linear Model 0.86581
```

2.3.3 Augmenting the Model with Additional Predictors

Building upon our regularized linear model, we now enhance it by including more predictors. This is a critical step towards refining the model's accuracy. We start by assessing the correlation of each potential predictor with the residuals from our existing model. The strongest correlated variables are chosen as new predictors, significantly enriching the model's capacity to capture complex patterns in the data.

```r
# Selecting potential predictors while excluding non-predictor columns
predictors <- model_data_train  |>
  select( c(residual, !c(userId, movieId, rating, b_user, b_movie, rating_hat)))

# Calculating the absolute correlation of each predictor with the residuals
corr <- abs(cor(predictors)[-1, 1])
```



Armed with insights from our correlation analysis, we now incorporate these new predictors into our regularized model. This advanced model, which now includes temporal dynamics and popularity factors, promises improved predictions by capturing the nuanced effects of time and user engagement.

This model is not just a mere combination of biases but a sophisticated prediction engine that understands the essence of user preferences and movie lifecycles. We fine-tune this model with a rigorous cross-validation procedure, carefully selecting the regularization parameter that harmonizes complexity with predictive accuracy.

The predictors are calculated as:

$$b_{\text{days}} = \frac{1}{n+\lambda} \sum_i (rating_i - \hat{\mu} - b_{\text{movie}} - b_{\text{user}}) = \sum_i \frac{rating_i - \hat{\mu} - b_{\text{movie}} - b_{\text{user}}}{n+\lambda}$$

and

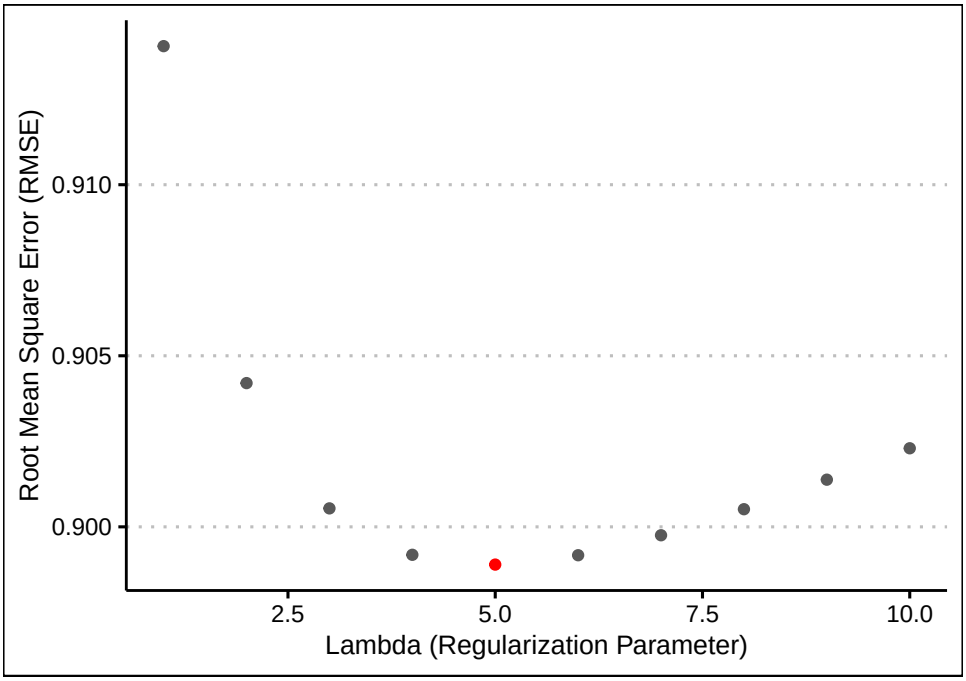
$$b_{\text{count}} = \frac{1}{n+\lambda} \sum_i (rating_i - \hat{\mu} - b_{\text{movie}} - b_{\text{user}} - b_{\text{days}}) = \sum_i \frac{rating_i - \hat{\mu} - b_{\text{movie}} - b_{\text{user}} - b_{\text{days}}}{n+\lambda}$$

The choice of a favorable λ is determined through a two-step cross-validation process using the smaller datasets *small_data_train* and *small_data_test* for efficiency. In the first step, the values 1, 2, ... 10 are tested for λ. Subsequently, in the second step, additional values in the vicinity of the initial λ are examined. This iterative approach ensures a thorough exploration of possible λ values.

```
# Rating as vector
rating <- small_data_test |> pull(rating)

# Execute cross-validation to determine the best lambda for the advanced model
train_advanced_linear_model <- function(lambda){

  # Compute additional biases based on days and count, incorporating regularization
  small_data_train <- small_data_train |>
    group_by(movieId) |>
    mutate(b_movie = sum(rating - mu_hat) / (n() + lambda)) |>
    ungroup() |>
    group_by(userId) |>
    mutate(b_user = sum(rating - mu_hat - b_movie) / (n() + lambda)) |>
    ungroup() |>
    group_by(days_since_movie_first_rating) |>
    mutate(b_days = sum(rating - mu_hat - b_movie - b_user) / (n() + lambda)) |>
    ungroup() |>
    group_by(movie_rating_count) |>
    mutate(b_count = sum(rating - mu_hat - b_movie - b_user - b_days) /
             (n() + lambda))

  # Save computed biases for use in prediction
  fit_movies <- small_data_train |>
    group_by(movieId) |>
    summarize(b_movie = first(b_movie))

  fit_users <- small_data_train |>
    group_by(userId) |>
    summarize(b_user = first(b_user))
```

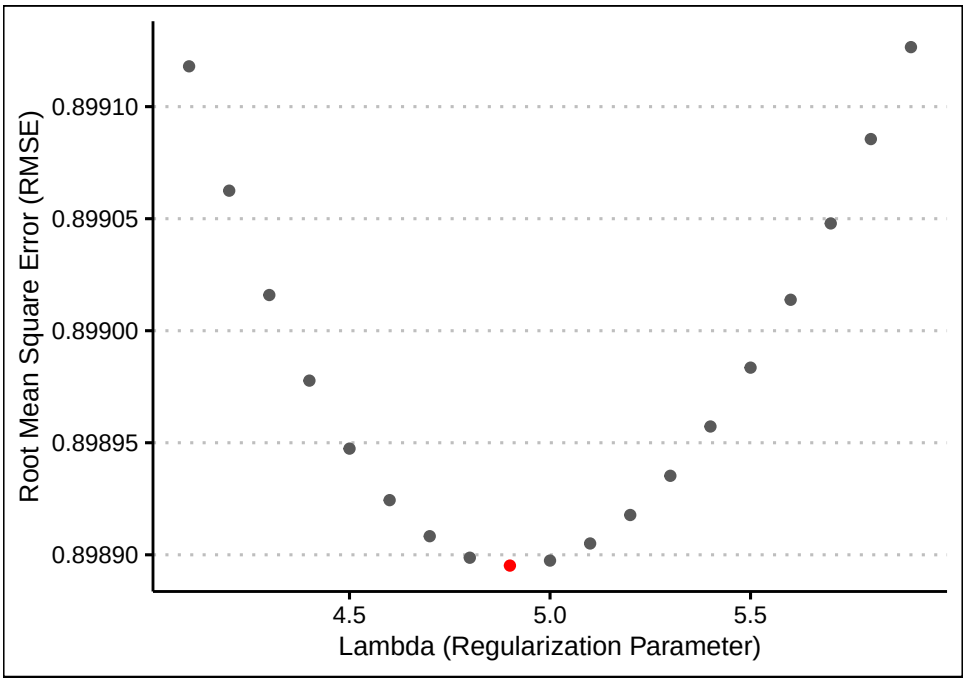
```
  fit_days <- small_data_train |>
    group_by(days_since_movie_first_rating) |>
    summarize(b_days = first(b_days))

  fit_count <- small_data_train |>
    group_by(movie_rating_count) |>
    summarize(b_count = first(b_count))

  # Predict ratings and calculate RMSE for the advanced model
  rating_hat <-
    small_data_test |>
    left_join(fit_movies, by = "movieId") |>
    left_join(fit_users, by = "userId") |>
    left_join(fit_days, by = "days_since_movie_first_rating")  |>
    left_join(fit_count, by = "movie_rating_count")  |>
    mutate(rating_hat = mu_hat + b_movie + b_user + b_days + b_count) |>
    pull(rating_hat)

  RMSE(rating, rating_hat)
}

# Initial search for the best lambda using a range of values
lambdas <- seq(1, 10, 1)
rmses <- sapply(lambdas, train_advanced_linear_model)
lambda <- lambdas[which.min(rmses)]
print(lambda)
```

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



```r
# Fine-tuning the lambda value by searching in a narrower range
lambdas <- seq(lambda - .9, lambda + .9, .1)
rmses <- sapply(lambdas, train_advanced_linear_model)
lambda <- lambdas[which.min(rmses)]
print(lambda)
```

```
## [1] 4.9
```



With our λ fine-tuned, we now train the advanced regularized model on the comprehensive dataset. This model integrates four pivotal factors: the movie's innate appeal, the user's bias, the influence of the movie's release recency, and the volume of ratings. Each factor is a cog in the mechanism that powers our prediction engine, and together, they work in concert to generate precise ratings predictions.

```r
# Calculate biases including new predictors for the advanced model
model_data_train <- model_data_train |>
  group_by(movieId) |>
  mutate(b_movie = sum(rating - mu_hat) / (n() + lambda)) |>
  ungroup() |>
  group_by(userId) |>
  mutate(b_user = sum(rating - mu_hat - b_movie) / (n() + lambda))  |>
  ungroup() |>
  group_by(days_since_movie_first_rating) |>
  mutate(b_days = sum(rating - mu_hat - b_movie - b_user) / (n() + lambda)) |>
  ungroup() |>
  group_by(movie_rating_count) |>
  mutate(
    b_count = sum(rating - mu_hat - b_movie - b_user - b_days) / (n() + lambda),
    rating_hat = mu_hat + b_movie + b_user + b_days + b_count,
    residual = rating - rating_hat
  ) |>
  ungroup()
```

```r
# Save calculated biases for prediction
fit_movies <- model_data_train |>
  group_by(movieId) |>
  summarize(b_movie = first(b_movie))

fit_users <- model_data_train |>
  group_by(userId) |>
  summarize(b_user = first(b_user))

fit_days <- model_data_train |>
  group_by(days_since_movie_first_rating) |>
  summarize(b_days = first(b_days))

fit_count <- model_data_train |>
  group_by(movie_rating_count) |>
  summarize(b_count = first(b_count))

# Predict ratings for the test dataset and calculate RMSE for the advanced model
rating_hat <- model_data_test |>
  left_join(fit_movies, by = "movieId")  |>
  left_join(fit_users, by = "userId")  |>
  left_join(fit_days, by = "days_since_movie_first_rating")  |>
  left_join(fit_count, by = "movie_rating_count")  |>
  mutate(rating_hat = mu_hat + b_movie + b_user + b_days + b_count) |>
  pull(rating_hat)

# Update evaluation data frame with the new RMSE
evaluation <- union(
  evaluation,
  data.frame(
    data = "model",
    model = "regularized linear: mu + b_movie + b_user + b_days + b_count",
    RMSE = RMSE(model_data_test$rating, rating_hat)
  )
)
evaluation
```

```
##    data                                                    model    RMSE
## 1 model                        simple linear: mu + b_movie + b_user 0.86638
## 2 model                                   Regularized Linear Model 0.86581
## 3 model regularized linear: mu + b_movie + b_user + b_days + b_count 0.86498
```

2.3.4 Developing Residual Models

Expanding our model to include different types of analysis, we explore additional models such as GLM, RF, and loess. These are particularly adept at utilizing all predictor variables available. To manage computation resources, we use smaller datasets for these models, focusing on modeling the residuals.

The residual is the discrepancy between actual ratings and those predicted by our current model. By targeting this, we can refine our predictions through a combination of diverse modeling techniques.

To address this, model training was conducted with a smaller dataset, varying in size based on each model's

requirements. To facilitate the integration of the linear model with these additional models, they were constructed for the residual:

$$\text{residual} = \text{rating} - \hat{\text{rating}}$$

This method ensures a cohesive combination of the linear model with the diverse set of additional models.

```
# Function to train residual models
train_residual_model <- function(model, data_set, p) {
  # generate smaller (p) sample from dataset for next model to avoid long training times
  set.seed(5)
  small_index <-
    createDataPartition(
      y = data_set$rating,
      times = 1,
      p = p,
      list = FALSE
    )
  small_data_train <- data_set[small_index, ]

  # Training the model with all predictors
  set.seed(6)
  train(
    residual ~ userId + movieId + days_since_movie_first_rating + movie_rating_count +
      movie_year + movie_age + rating_year +
      genre_Drama + genre_Comedy + genre_Action + genre_Thriller + genre_Adventure +
      days_since_user_first_rating + user_rating_count,
    method = model,
    data = small_data_train
  )
}
```

Initially, a General Linear Model (GLM) for the residual is trained, incorporating all available variables. The predictions are calculated by combining the prediction for the residual with the prediction from the linear model:

$$\text{final } \hat{\text{rating}} = \hat{\text{rating}} + \hat{\text{residual}}$$

This dual-model approach aims to provide a more nuanced understanding of the data by integrating both linear and GLM perspectives into the modeling process. The resulting Root Mean Square Error (*RMSE*) is saved in *evaluation*.

```
# Train a GLM model on residuals with the relevant predictors
train_glm <- train_residual_model("glm", model_data_train, .05)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, : There were missing v
```

```
# predict residuals with GLM model
predict_glm <- predict(train_glm, model_data_test)

# final prediction as advanced linear model + residual model
final_rating_hat <- rating_hat + predict_glm
```

```r
# Record result
evaluation <- union(
  evaluation,
  data.frame(
    data = "model",
    model = "regularized linear + GLM for residual",
    RMSE = RMSE(model_data_test$rating, final_rating_hat)
  )
)
evaluation
```

```
##    data                                                        model    RMSE
## 1 model                          simple linear: mu + b_movie + b_user 0.86638
## 2 model                                       Regularized Linear Model 0.86581
## 3 model regularized linear: mu + b_movie + b_user + b_days + b_count 0.86498
## 4 model                         regularized linear + GLM for residual 0.86493
```

2.3.5 Creating an Ensemble of Residual Models

To enhance our predictive accuracy, we construct an ensemble model combining Random Forest (RF) and Local Regression (loess) models. This ensemble approach aims to leverage the unique strengths of each model, creating a more robust prediction system.

The ensemble predictions are obtained using a weighted average approach, where weights are determined by the sample sizes used in training each model. This method balances the contribution of each model based on its training breadth.

```r
# Two more models for ensemble
# Handling potential warnings during model training
suppressWarnings({
  # Train RF and loess models using smaller subsets for efficiency
  train_rf <- train_residual_model("rf", model_data_train, .0001)
  train_loess <- train_residual_model("gamLoess", model_data_train, .002)

  # Predict residuals using both RF and loess models
  predict_rf <- predict(train_rf, model_data_test)
  predict_loess <- predict(train_loess, model_data_test)

})
```

The final prediction integrates the advanced linear model with the ensemble residuals, resulting in a comprehensive prediction formula. The performance of this ensemble model, measured by Root Mean Square Error (*RMSE*), is then recorded for comparison and evaluation.

```r
# Calculating the ensemble prediction using weights derived from training sample sizes
final_rating_hat <-
  rating_hat + (.05 * predict_glm + .0001 * predict_rf + .002 * predict_loess) /
               (.05 + .0001 + .002)

# Recording the ensemble model's performance
evaluation <- union(
  evaluation,
```

```
  data.frame(
    data = "model",
    model = "regularized linear + ensemble for residual",
    RMSE = RMSE(model_data_test$rating, final_rating_hat)
  )
)
evaluation
```

```
##    data                                                        model    RMSE
## 1 model                          simple linear: mu + b_movie + b_user 0.86638
## 2 model                                       Regularized Linear Model 0.86581
## 3 model regularized linear: mu + b_movie + b_user + b_days + b_count 0.86498
## 4 model                         regularized linear + GLM for residual 0.86493
## 5 model                  regularized linear + ensemble for residual 0.86492
```

The ensemble model, blending various statistical and machine learning techniques, represents a significant stride in our quest for accurate movie rating predictions. It showcases our ability to integrate diverse methodologies into a single, cohesive prediction tool.

3 Results: Analysis and Validation

In this crucial chapter, we rigorously evaluate our array of developed models using the dedicated *final_holdout_test* dataset. This dataset, distinct from the one used in model development, allows us to objectively assess each model's predictive prowess on unseen data.

The validation process encompasses a spectrum of models, starting from the foundational simple linear model and extending to the more intricate regularized and ensemble models. The critical parameter λ, identified during the model construction phase, plays a vital role in the validation of the regularized models.

This comprehensive validation approach is key to understanding each model's strengths and limitations, guiding us towards the most effective solution for movie rating prediction.

3.1 Validation of the Simple Linear Model

The initial phase of our validation process involves the simple linear model, characterized by the formula

$$\hat{y} = \hat{\mu} + b_{\text{movie}} + b_{\text{user}} + \text{error}$$

This model, originally built using the *edx* dataset, now faces the ultimate test against the *final_holdout_test* dataset. The outcomes of this validation are meticulously recorded, providing us with initial insights into the model's effectiveness in real-world scenarios.

```
# Construct the simple linear model using the edx dataset
mu_hat <- mean(edx$rating)

# Compute movie and user biases
edx <- edx |>
  group_by(movieId) |>
  mutate(b_movie = mean(rating - mu_hat)) |>
  ungroup() |>
  group_by(userId) |>
```

```
  mutate(b_user = mean(rating - mu_hat - b_movie)) |>
  ungroup()

# Store biases for model validation
fit_movies <- edx |>
  group_by(movieId) |>
  summarize(b_movie = first(b_movie))

fit_users <- edx |>
  group_by(userId) |>
  summarize(b_user = first(b_user))

# Apply the model to the final holdout test dataset for validation
rating_hat_simple <- final_holdout_test |>
  left_join(fit_movies, by = "movieId")  |>
  left_join(fit_users, by = "userId")  |>
  mutate(rating_hat = mu_hat + b_movie + b_user) |>
  pull(rating_hat)

# Evaluate and record the model's performance
evaluation <- union(
  evaluation,
  data.frame(
    data = "validate",
    model = "simple linear: mu + b_movie + b_user",
    RMSE = RMSE(final_holdout_test$rating, rating_hat_simple)
  )
)
evaluation
```

```
##       data                                                          model    RMSE
## 1    model                        simple linear: mu + b_movie + b_user 0.86638
## 2    model                                    Regularized Linear Model 0.86581
## 3    model regularized linear: mu + b_movie + b_user + b_days + b_count 0.86498
## 4    model                       regularized linear + GLM for residual 0.86493
## 5    model                 regularized linear + ensemble for residual 0.86492
## 6 validate                        simple linear: mu + b_movie + b_user 0.86535
```

3.2 Regularized Simple Linear Model

In this step, the regularized linear model is built using the *edx* and validated with *final_holdout_test*. $\lambda = 5$ is applied for the model. The resulting *RSME* is saved in *evaluation*.

```
# Use lambda from before
lambda <- 5

# Train model, compute b_movie, b_user
edx <- edx |>
  group_by(movieId) |>
  mutate(b_movie = sum(rating - mu_hat) / (n() + lambda)) |>
  ungroup() |>
  group_by(userId) |>
```

```
  mutate(b_user = sum(rating - mu_hat - b_movie) / (n() + lambda))  |>
  ungroup()

# save b_movie, b_user
fit_movies <- edx |>
  group_by(movieId) |>
  summarize(b_movie = first(b_movie))

fit_users <- edx |>
  group_by(userId) |>
  summarize(b_user = first(b_user))

# Compute prediction
rating_hat <- final_holdout_test |>
  left_join(fit_movies, by = "movieId")  |>
  left_join(fit_users, by = "userId")  |>
  mutate(rating_hat = mu_hat + b_movie + b_user ) |>
  pull(rating_hat)

# Record result
evaluation <- union(
  evaluation,
  data.frame(
    data = "validate",
    model = "regularized linear: mu + b_movie + b_user",
    RMSE = RMSE(final_holdout_test$rating, rating_hat)
  )
)
evaluation
```

```
##       data                                                          model    RMSE
## 1    model                          simple linear: mu + b_movie + b_user 0.86638
## 2    model                                      Regularized Linear Model 0.86581
## 3    model regularized linear: mu + b_movie + b_user + b_days + b_count 0.86498
## 4    model                         regularized linear + GLM for residual 0.86493
## 5    model                   regularized linear + ensemble for residual 0.86492
## 6 validate                          simple linear: mu + b_movie + b_user 0.86535
## 7 validate                    regularized linear: mu + b_movie + b_user 0.86482
```

3.3 Validation of the Regularized Simple Linear Model

The next phase in our model validation journey involves the regularized simple linear model. This model, which incorporates a regularization parameter, $\lambda = 5$, aims to refine our prediction accuracy by balancing the bias terms, especially in cases with limited data. The *edx* dataset is utilized to train this model, and the *final_holdout_test* dataset serves as the testing ground to gauge its performance in a real-world scenario.

This validation process is pivotal, as it allows us to assess the impact of regularization on our model's effectiveness and accuracy. The findings from this step are instrumental in guiding our decision-making for potential model adjustments or enhancements.

```
# Applying the previously determined lambda value
lambda <- 4.9
```

```r
# Building the regularized model with edx dataset
edx <- edx |>
  group_by(movieId) |>
  mutate(b_movie = sum(rating - mu_hat) / (n() + lambda)) |>
  ungroup() |>
  group_by(userId) |>
  mutate(b_user = sum(rating - mu_hat - b_movie) / (n() + lambda))  |>
  ungroup() |>
  group_by(days_since_movie_first_rating) |>
  mutate(b_days = sum(rating - mu_hat - b_movie - b_user) / (n() + lambda)) |>
  ungroup() |>
  group_by(movie_rating_count) |>
  mutate(b_count = sum(rating - mu_hat - b_movie - b_user - b_days) / (n() + lambda),
         rating_hat = mu_hat + b_movie + b_user + b_days + b_count,
         residual = rating - rating_hat) |>
  ungroup()


# Storing the bias terms for validation
fit_movies <- edx |>
  group_by(movieId) |>
  summarize(b_movie = first(b_movie))

fit_users <- edx |>
  group_by(userId) |>
  summarize(b_user = first(b_user))

fit_days <- edx |>
  group_by(days_since_movie_first_rating) |>
  summarize(b_days = first(b_days))

fit_count <- edx |>
  group_by(movie_rating_count) |>
  summarize(b_count = first(b_count))

# Validating the model using final holdout test dataset
rating_hat <- final_holdout_test |>
  left_join(fit_movies, by = "movieId")  |>
  left_join(fit_users, by = "userId")  |>
  left_join(fit_days, by = "days_since_movie_first_rating")  |>
  left_join(fit_count, by = "movie_rating_count")  |>
  mutate(rating_hat = mu_hat + b_movie + b_user + b_days + b_count) |>
  pull(rating_hat)

# Recording the model's performance
evaluation <- union(
  evaluation,
  data.frame(
    data = "validate",
    model = "regularized linear: mu + b_movie + b_user + b_days + b_count",
    RMSE = RMSE(final_holdout_test$rating, rating_hat)
  )
)
```

```
evaluation
```

```
##       data                                                          model    RMSE
## 1    model                          simple linear: mu + b_movie + b_user 0.86638
## 2    model                                       Regularized Linear Model 0.86581
## 3    model regularized linear: mu + b_movie + b_user + b_days + b_count 0.86498
## 4    model                         regularized linear + GLM for residual 0.86493
## 5    model                  regularized linear + ensemble for residual 0.86492
## 6 validate                          simple linear: mu + b_movie + b_user 0.86535
## 7 validate                   regularized linear: mu + b_movie + b_user 0.86482
## 8 validate regularized linear: mu + b_movie + b_user + b_days + b_count 0.86380
```

3.4 Residual Models Analysis

In our final modeling stage, we explore residual models using General Linear Model (GLM), Random Forest (RF), and Local Regression (loess). These models are trained on a scaled-down version of the *edx* dataset, focusing on the residual aspect of the ratings.

The use of GLM, RF, and loess models is a strategic choice, each bringing unique strengths to our predictive framework. This diversified approach allows us to capture different facets of the data that might be overlooked by a single model.

The process starts with GLM, followed by an ensemble method that combines GLM with RF and loess. This ensemble approach aims to leverage the collective power of these models for superior prediction accuracy.

```
# Training the GLM model
train_glm <- train_residual_model("glm", edx, .05)

# Predicting residuals using the trained GLM model
predict_glm <- predict(train_glm, final_holdout_test)

# Integrating GLM predictions with the advanced linear model
final_rating_hat <- rating_hat + predict_glm

# Documenting the validation results
evaluation <- union(
  evaluation,
  data.frame(
    data = "validate",
    model = "regularized linear + GLM for residual",
    RMSE = RMSE(final_holdout_test$rating, final_rating_hat)
  )
)
evaluation
```

```
##       data                                                          model    RMSE
## 1    model                          simple linear: mu + b_movie + b_user 0.86638
## 2    model                                       Regularized Linear Model 0.86581
## 3    model regularized linear: mu + b_movie + b_user + b_days + b_count 0.86498
## 4    model                         regularized linear + GLM for residual 0.86493
## 5    model                  regularized linear + ensemble for residual 0.86492
## 6 validate                          simple linear: mu + b_movie + b_user 0.86535
## 7 validate                   regularized linear: mu + b_movie + b_user 0.86482
```

```
## 8 validate regularized linear: mu + b_movie + b_user + b_days + b_count 0.86380
## 9 validate                          regularized linear + GLM for residual 0.86371
```

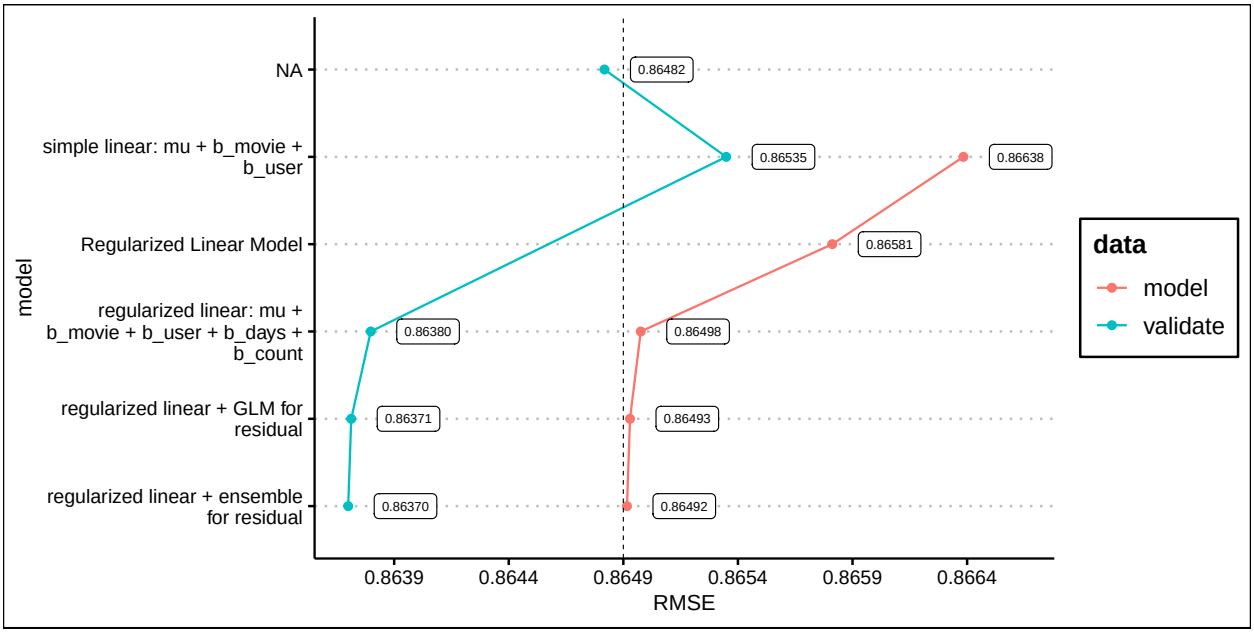
```
# Suppressing warnings for RF and loess model training
suppressWarnings({

  # Training RF and loess models with different sample sizes
  train_rf <- train_residual_model("rf", edx, .0001)
  train_loess <- train_residual_model("gamLoess", edx, .002)

  # Predicting residuals with RF and loess models
  predict_rf <- predict(train_rf, final_holdout_test)
  predict_loess <- predict(train_loess, final_holdout_test)

})


# Creating an ensemble prediction combining GLM, RF, and loess residuals
final_rating_hat <-
  rating_hat + (.05 * predict_glm + .0001 * predict_rf + .002 * predict_loess) /
               (.05 + .0001 + .002)

# Recording the ensemble model's validation results
evaluation <- union(
  evaluation,
  data.frame(
    data = "validate",
    model = "regularized linear + ensemble for residual",
    RMSE = RMSE(final_holdout_test$rating, final_rating_hat)
  )
)
evaluation
```

```
##        data                                                          model    RMSE
## 1     model                          simple linear: mu + b_movie + b_user 0.86638
## 2     model                                      Regularized Linear Model 0.86581
## 3     model regularized linear: mu + b_movie + b_user + b_days + b_count 0.86498
## 4     model                         regularized linear + GLM for residual 0.86493
## 5     model                  regularized linear + ensemble for residual 0.86492
## 6  validate                           simple linear: mu + b_movie + b_user 0.86535
## 7  validate                    regularized linear: mu + b_movie + b_user 0.86482
## 8  validate regularized linear: mu + b_movie + b_user + b_days + b_count 0.86380
## 9  validate                         regularized linear + GLM for residual 0.86371
## 10 validate                  regularized linear + ensemble for residual 0.86370
```

3.5 Conclusion

The following graph presents a comparative analysis of the RMSE across different models during the construction and validation phases. Noticeable improvements are evident with the incorporation of regularization and additional predictors in the linear model. The residual models, while beneficial, show a lesser degree of impact, possibly limited by the sample size constraints. Variations in model accuracy between the construction and validation phases highlight the influence of using the complete *edx* dataset for training in the validation stage.

This visual representation facilitates a clear understanding of each model's performance and the effectiveness of various enhancements applied throughout the project.



4 Discussion

The project culminates with a comprehensive model that effectively meets the RMSE target set for the course. This success is attributed to the evolution of the initial simple linear model into a sophisticated predictive framework, enriched by additional features such as movie ratings count and movie age.

A key factor in enhancing accuracy is the strategic use of regularization, impacting both the basic and advanced linear models. Despite computational constraints, employing models like GLM, RF, and loess on a subset of data showcased their potential in refining prediction accuracy.

The project, while focused on its current methodology, hints at further avenues for improvement. The integration of genre-based variables, particularly in drama, could offer further precision. However, this approach remains closely tied to the existing methods without introducing significant novelty.

Future exploration might include matrix factorization techniques, promising even better outcomes. However, constraints related to the course scope and available resources led to this possibility remaining unexplored in the current project.