Movielens Recommender System

Matthias Frye

2023-12-13

Contents

1 Introduction

This project was conducted as part of the HarvardX Data Science Capstone Module and was inspired by the Netflix challenge. The primary goal is to develop a movie recommendation system that leverages the extensive Movielens dataset of 10 million entries provided by the Department of Computer Science and Engineering at the University of Minnesota.

The Movielens database comprises movies, complete with title and genre information, alongside movie ratings that include user and timestamp details. The overarching objective involves constructing a predictive model for unknown ratings. The model's accuracy is assessed using the Root Mean Square Error ($RMSE$) metric.

2 Method

This chapter outlines the data preparation process, the construction of models, and the subsequent evaluation.

2.1 Overview

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

1. Data Load, Analyis and Preparation

The Movielens datasets are loaded from the files supplied in the course, initiating a sequence of straightforward analyses aimed at comprehending the dataset. To facilitate the implementation of advanced models, supplementary variables used as predictors are computed and presented visually. At last, the data is partinioned into model and validation datasets. The model dataset is further partinioned into a train and test dataset and a small dataset is created to avoid long run times in some of the later steps.

2. Model Development

Multiple prediction models are constructed and tested using subsets of the model data. The progression involves enhancing the approach step by step:

- Initially, a simple linear model is established, employing movie and user as the sole predictors.
- The simple linear model undergoes refinement through the incorporation of regularization.
- A more intricate regularized linear model is crafted by introducing two additional predictors: the number of ratings for a movie and the duration since the movie was first rated.
- A general linear model is devised for the residuals of the preceding model.
- An ensemble of models is created for the residuals of the aforementioned model.

3. Model Validation

Model validation was executed using a distinct subset of the data, exclusively reserved for this purpose.

2.2 Data Load, Analysis and Preparation

Initially, required libraries are loaded:

- *tidyverse* is a collection of packages for data representation and management
- *caret* provides a number of functions for classification and regression

- *doParallel* enables and controls parallel processing
- *timeDate* provides functions for handling timedate data
- *stringi* provides fast string functions
- *gam* offers generalized additive models
- *scales* provides functions for formatting axes in graphs
- *ggthemes* offers a number of nice looking graph themes for *ggplot*

```
library(tidyverse)
library(caret)
library(doParallel)
library(timeDate)
library(stringi)
library(gam)
library(scales)
library(ggthemes)
```

2.2.1 Data Load

Subsequently, the Movielens data is loaded. This code was provided in the course instructions.

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

# Note: this process could take a couple of minutes

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")
```

2.2.2 Data Analysis

In the upcoming section, a comprehensive analysis of the data and rating distribution unfolds. The exploration commences with an exposition of the data structure, accompanied by the calculation of the number of unique movies and users. Subsequently, the mean rating is computed.

```
# understand structure of the data
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"
```

```
# calculate the number of distinct movies and users
n_distinct(movielens$movieId)
```

```
## [1] 10677
```

```
n_distinct(movielens$userId)
```

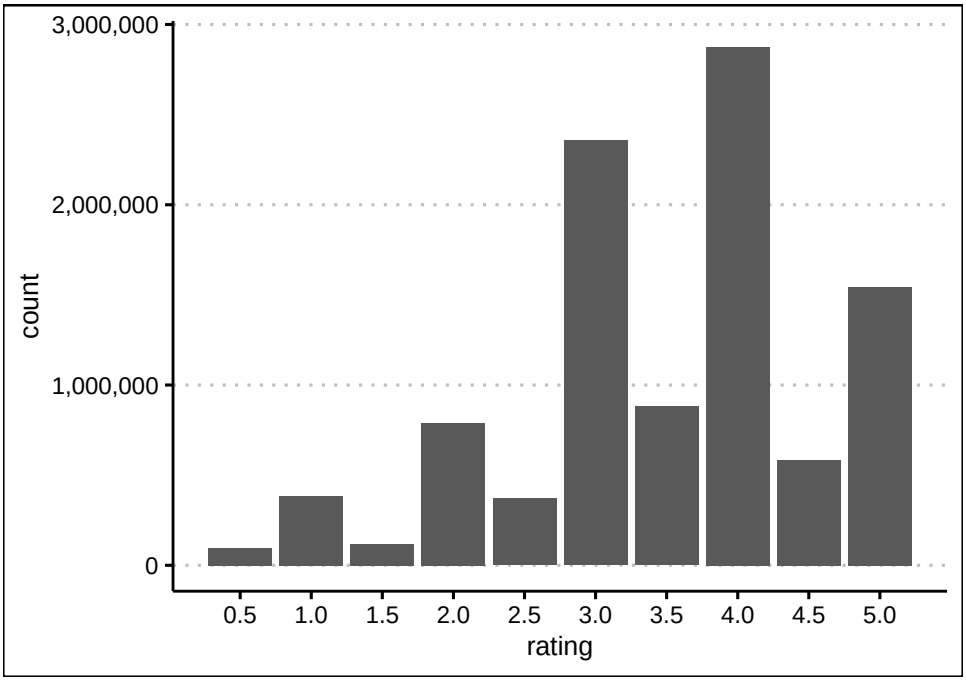
```
## [1] 69878
```

As the ratings are most critical for the project, the ratings are investigated further. The average rating is calculated.

```
# average rating
mean(movielens$rating)
```

```
## [1] 3.512422
```

The following histogram depicts the distribution of ratings and provides insights into the overall pattern. It's interesting to observe that the distribution of ratings leans towards whole numbers rather than half stars.



Well known movies have thousands of ratings from different users as the following analysis of the top 5 movies with most ratigs shows. The rating for theses most rated movies is above average.

```
## # 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)
```

In contrast, the top 5 rated movies are less well known and have only received a small number of ratings.

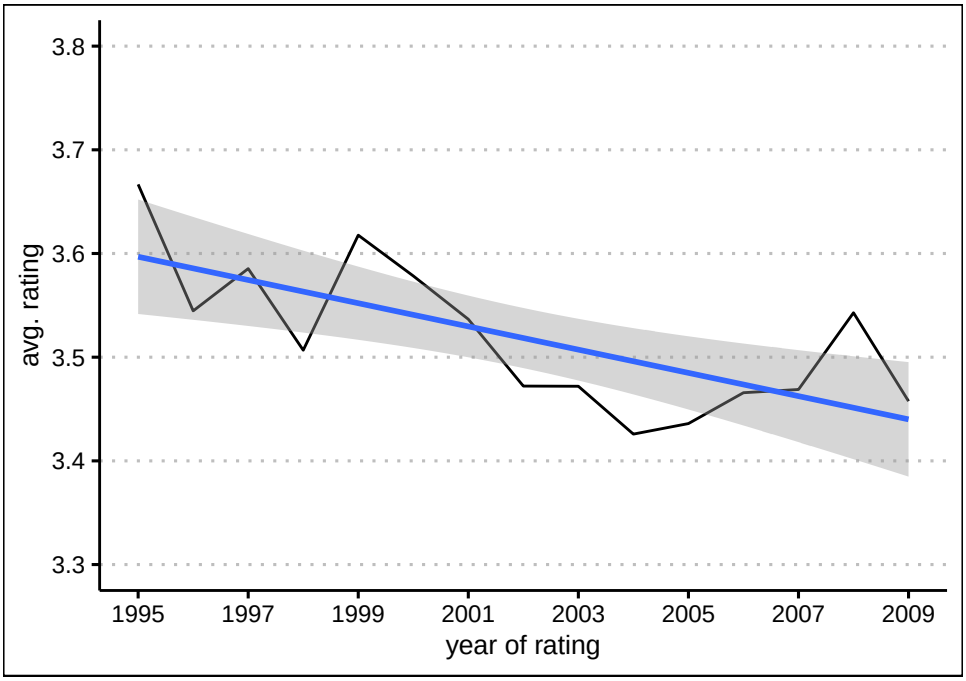
```
## # A tibble: 5 x 4
##   movieId count   avg title
##     <int> <int> <dbl> <chr>
## 1   33264     2     5 Satan's Tango (Sátántangó) (1994)
## 2   42783     1     5 Shadows of Forgotten Ancestors (1964)
## 3   51209     1     5 Fighting Elegy (Kenka erejii) (1966)
## 4   53355     1     5 Sun Alley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

2.2.3 Data Preparation

In order to build advanced models, several additional variables are computed and integrated into the dataset. These calculations are executed prior to the segmentation of the data into model and validation sets. The resulting variables will be used as additional predictors in the models.

The date and year of the rating is added and the average rating by year is visualized. There is a clear trend towards lower ratings over time.

```r
movielens <- movielens |>
  mutate(rating_date = as.Date(as.POSIXct(timestamp, origin = "1970-01-01")),
         rating_year = year(rating_date))
```



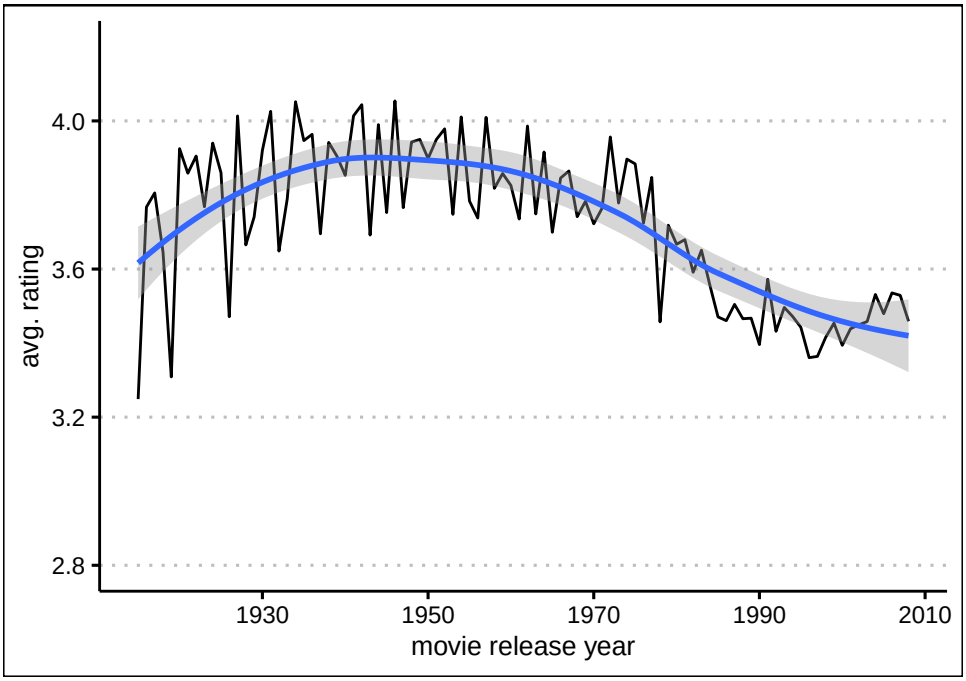
Release Year

The release year of the movies is embedded in the title in the format (yyyy) and requires extraction using a regular expression. A test is conducted to ensure that all titles conform to this pattern.

Following this, a graphical representation is created to show the impact of the release year on the average rating. Generally, older movies tend to receive higher ratings, with the notable exception of films released in the first half of the last century.

Additionally, the age of each movie at the time of rating is computed.

```r
# Pattern to extract movie release year in the format (yyyy) at the end of movie title
pattern <- "\\(\\d{4}\\)$"
# check whether all titles show the pattern
if (0 < sum(!str_detect(movielens$title, pattern)))
  stop ("Error extracting years from movie title")

# add release year of the movie and movie age at date of rating,
# use substr to remove "(" ")"
movielens <- movielens |>
  mutate(movie_year = as.integer(substr(str_match(title, pattern), 2, 5)),
         movie_age = rating_year - movie_year)
```

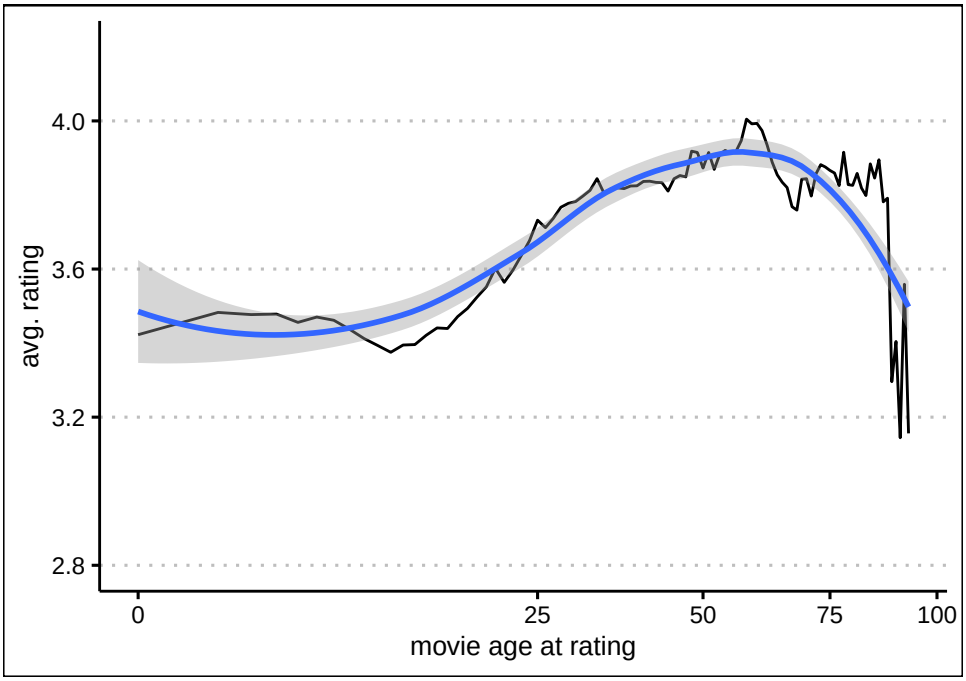


After calculation of the age of each movie at the time of rating, a verification process to ascertain whether all ratings occured subsequent to the movie's release is performed. The age is set to 0 for all exceptions.

```
# check rating date before release year
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.
```

Upon examining the influence of the movie's age at the time of rating, a parallel effect to that observed with the release year becomes apparent. Older movies generally receive higher ratings, albeit with exceptions in the case of very old films.



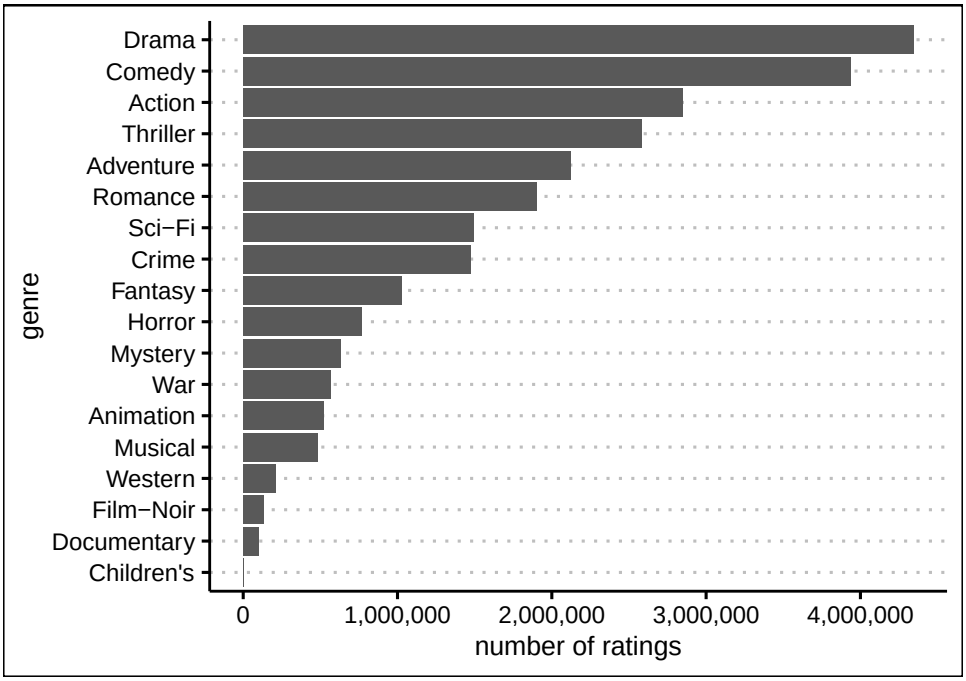
Genres

The next step involves an exploration of movie genres. Each movie can be associated with more than one genre, and the values in the genre variable have the format *"genre1|genre2|..."*.

According to the Movielens documentation, the following genres are present in the data: Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, and Western.

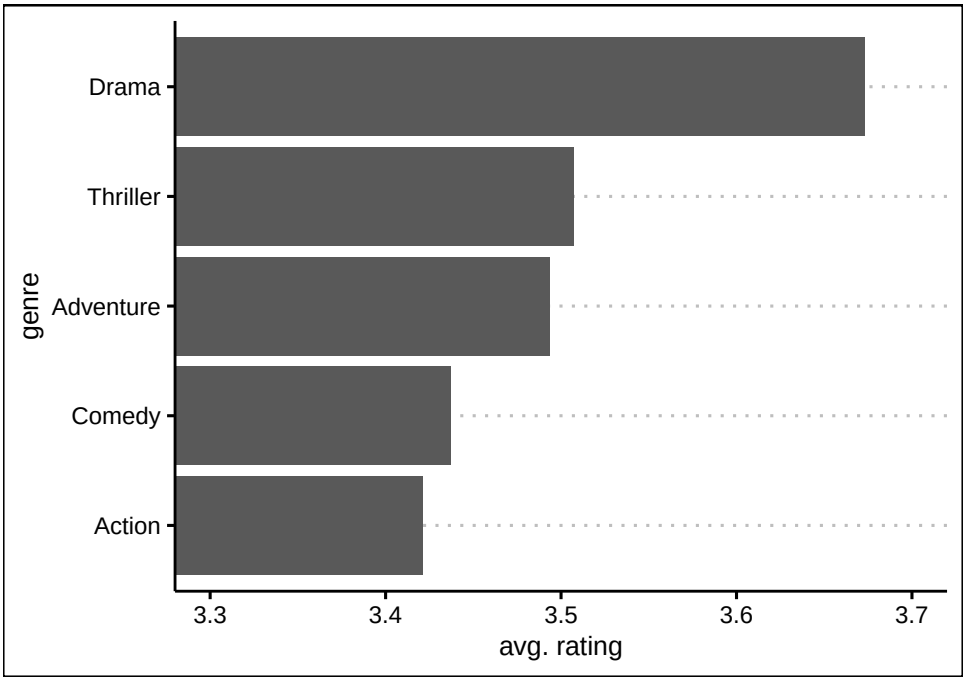
The top five genres are Drama, Comedy, Action, Thriller, and Adventure. To facilitate modeling, dichotomous variables were created for each genre, taking on values of 0 or 1. A value of 1 indicates that the movie is affiliated with the specific genre. For the sake of simplicity, these calculations were only carried out for the five most prevalent genres.

```
# 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")
# stri_detect_fixed is very fast
genre_counts <- sapply(all_genres, function(genre) {
  sum(stri_detect_fixed(movielens$genres, genre))
})

genre_counts <- data.frame(genre_counts = genre_counts,
                           genre = fct_reorder(labels(genre_counts), genre_counts))
```




```
# build top 5 genres as predictors
top5_genres <-
  genre_counts |>
  arrange(desc(genre_counts)) |>
  head(5) |>
  pull(genre)

# for each of top 5 genres: build genre predictor "is_genre" and rename to genre_xxx
for (i in 1:5) {
  movielens$is_genre <- stri_detect_fixed(movielens$genres, top5_genres[i])
  movielens$is_genre <- as.integer(movielens$is_genre)
  names(movielens)[names(movielens) == "is_genre"] <-
    paste("genre_", top5_genres[i], sep = "")
}
```

The new genre variables can be used to visualize the average rating for each of the top 5 genres.



Additional Date-related Variables

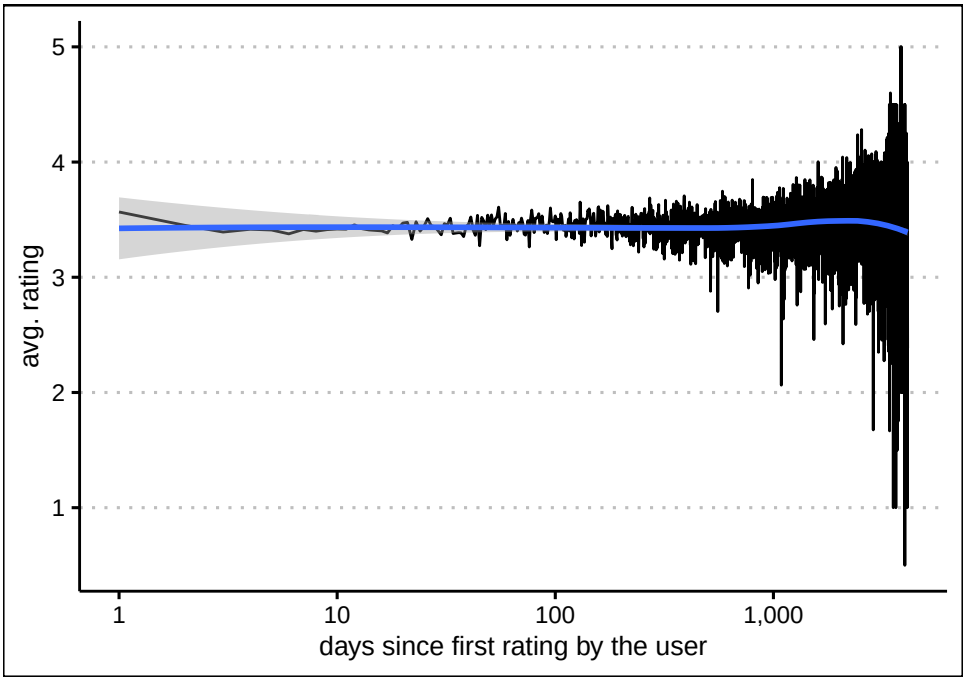
In an effort to further enhance modeling capabilities, additional date-related variables are computed and integrated into the Movielens dataset.

For each user, the date of their first movie rating is determined, and for subsequent ratings by the user, the temporal gap between each rating and the initial rating is computed. Additionally, the total number of ratings by a user is calculated.

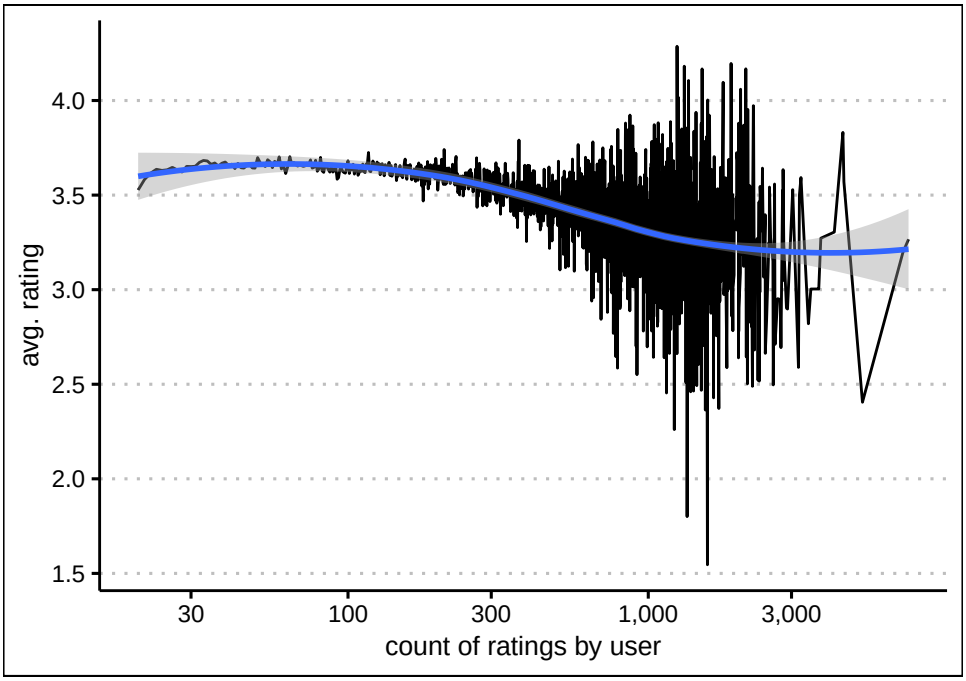
Similarly, the date of a movie's first rating is ascertained, and the difference between subsequent ratings and the initial rating is calculated. These supplementary variables provide valuable context to the dataset, facilitating more sophisticated modeling.

The accompanying charts illustrate that, for both movies and users, the temporal gap and rating counts significantly impact the average rating.

```
# date of first ratings of the user
movielens <- movielens |>
  group_by(userId) |>
  mutate(user_first_rating = min(rating_date))

# days between first user rating and current rating "rating experience"
movielens$days_since_user_first_rating <-
  as.integer(movielens$rating_date - movielens$user_first_rating) + 1
```

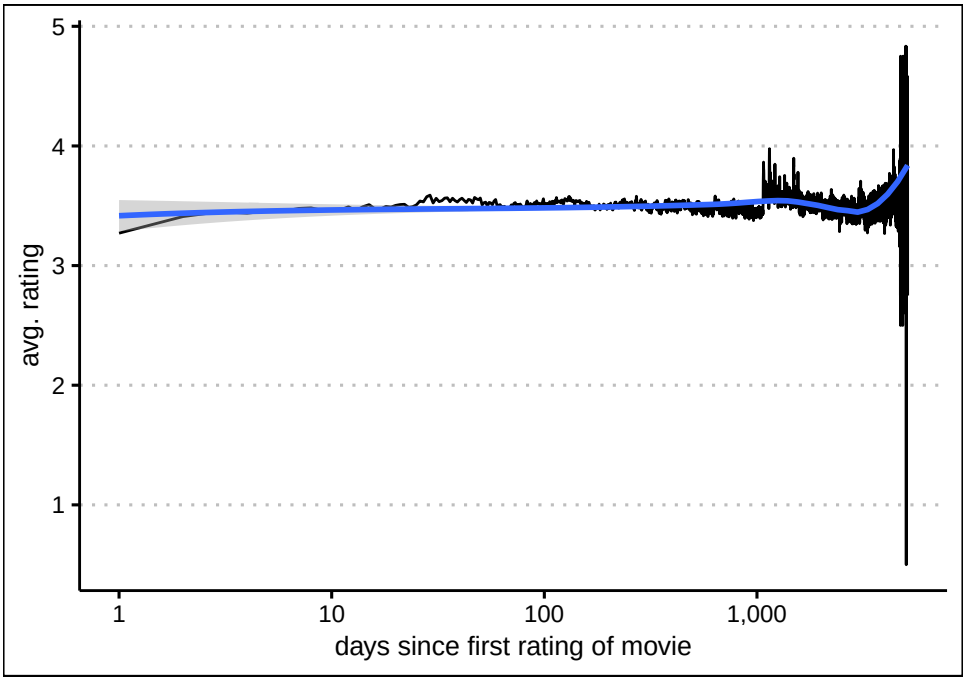



```
# add number of ratings by the user
movielens <- movielens |>
  group_by(userId) |>
  mutate(user_rating_count = n())
```

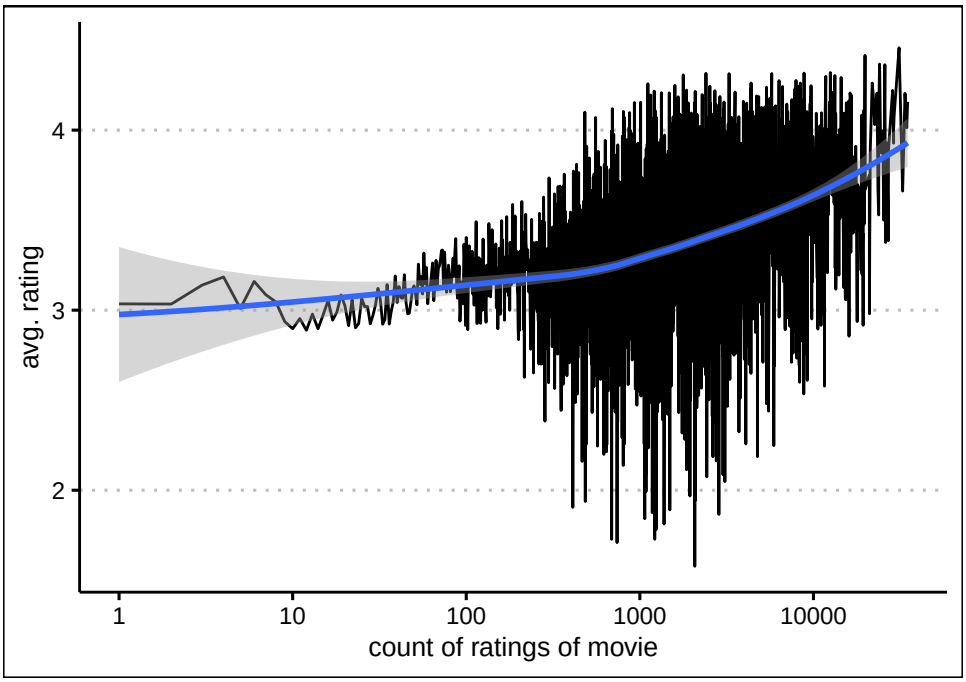



```
# add ate of first rating of the movie
movielens <- movielens |> group_by(movieId) |>
  mutate(movie_first_rating = min(rating_date))
# days between first movie rating and current rating "movie rating duration"
```

```r
movielens$days_since_movie_first_rating <-
  as.integer(movielens$rating_date - movielens$movie_first_rating) + 1
```




```r
# add number of ratings per movie
movielens <- movielens |> group_by(movieId) |>
  mutate(movie_rating_count = n())
```



2.2.4 Data Split

The data is now is split into model and validation data. This code was provided by the course and was not changed.

```
#### 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)
```

Training and Testing Datasets

In addition to segregating the data into model and validation datasets, we further divide the model dataset *edx* into two partitions—training and testing datasets. The training dataset encompasses 90%, while the testing dataset retains 10% of the data. This reserves the *final_holdout_test* data for the final validation.

```
# model_data_test holds 10% of edx for testing, 90% go into model_data_train
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, ]

# Make sure that model_test contains only movies, users, days_since_movie_first_rating and
# movie_rating_count with an entry in model_train
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")
```

Small Dataset

To avoid long processing times for regularization, an additional smaller (5%) sample is generated from *edx*. This small dataset is also split into a train and a test datasets.

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

```r
# 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 Model Development

Starting with a straightforward linear model, resembling the one covered in the course, a sequence of prediction models is methodically developed, advancing step by step and utilizing the *edx* dataset. The process begins with the simple linear model, followed by a regularized simple model. Subsequently, a more advanced linear model is constructed. Finally, the advanced linear model is synergistically combined with an additional General Linear Model (GLM), Random Forest (RF) Model, and a Local Regression (loess) model.

Next, parallel processing is enabled and the *RMSE* function is defined.

```r
# use half of the cores of local machine - optimum depends on number of cores
# and available physical memory
registerDoParallel(cores=detectCores()/2)

# check fice digits for RSME
options(digits = 5)

# RSME function to measure quality of model
RMSE <- function(true_ratings, predicted_ratings) {
  sqrt(mean((true_ratings - predicted_ratings) ^ 2))
}
```

2.3.1 Simple Linear Model

In this chapter, we establish the simple linear model, based on the formula

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

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)

# Record result
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 Regularized Simple Linear Model

Following the course methodology, the simple linear model undergoes expansion into a regularized linear model. To decrease the impact of ratings for movies with few ratings, the movie bias is computed differently. Instead of:

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

a factor λ is introduced:

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

Similarly, for the user bias, instead of:

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

it is calculated with a factor λ as:

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

To ensure a reasonable runtime, the selection of a favorable λ is determined through cross-validation using the smaller datasets *small_data_train* and *small_data_test*.

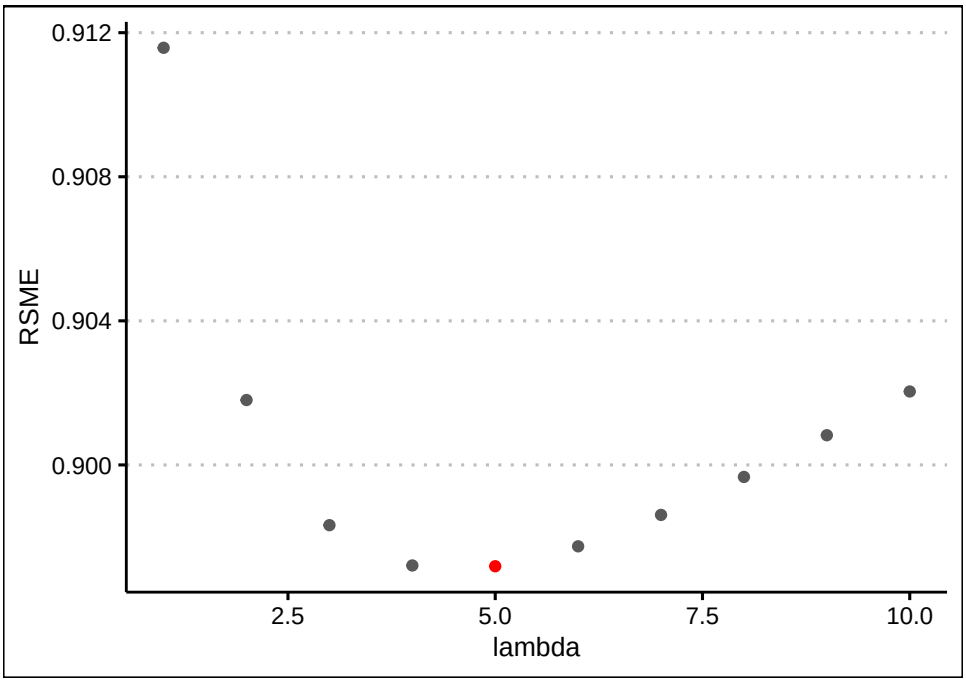
```
# True rating as vector
rating <- small_data_test |> pull(rating)

# use cross-validation to find lambda
train_simple_linear_model <- function(lambda){

  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()

  # Test model, save b_movie and b_user
  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))

  #Compute prediction
  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)
}

# try 10 lambdas
lambdas <- seq(1, 10, 1)
rmses <- sapply(lambdas, train_simple_linear_model)
lambda <- lambdas[which.min(rmses)]
print(lambda)
```

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

The following chart shows the Root Mean Square Error (*RMSE*) for different λ.

```
## Warning: The ‘<scale>‘ argument of ‘guides()‘ cannot be ‘FALSE‘. Use "none" instead as of ggplot2 3.3
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was generated.
```



With the computed λ from the previous step, the regularized model is trained using the larger dataset *model_data*. The residuals are defined as difference between true rating and predicted rating:

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

These residuals are stored in the dataset *model_data_train* for subsequent use and the resulting Root Mean Square Error (*RMSE*) is saved in *evaluation*.

```
model_data_train <- model_data_train |>
  group_by(movieId) |>
  mutate(b_movie = sum(rating - mu_hat)/(n() + lambda)) |>
  ungroup() |>
  group_by(userId) |>
```

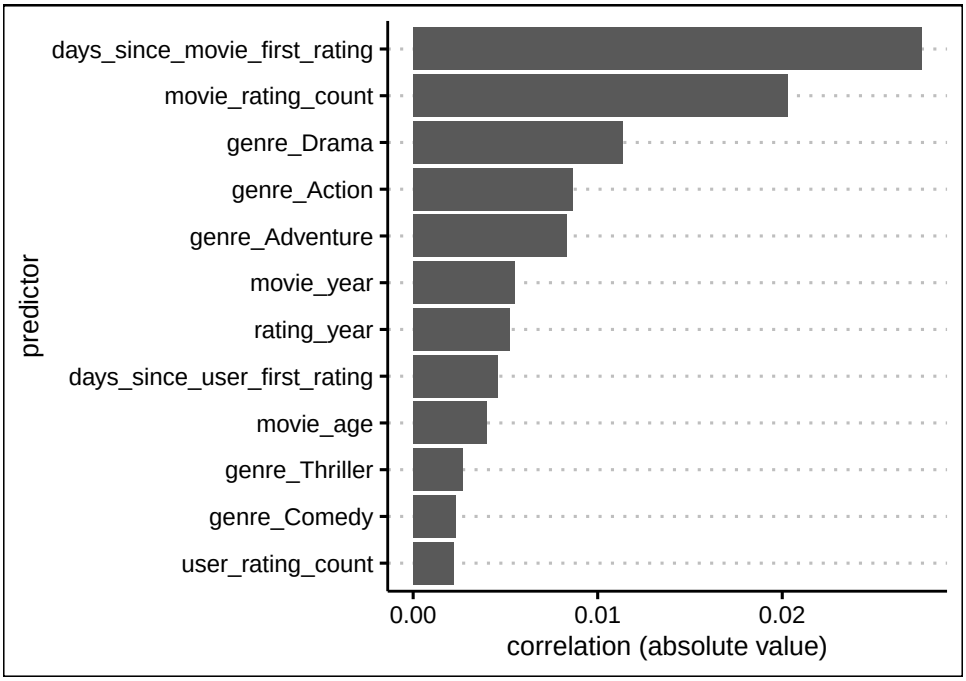
```
  mutate(b_user = sum(rating - mu_hat - b_movie)/(n() + lambda),
         rating_hat = mu_hat + b_movie + b_user,
         residual = rating - rating_hat) |>
  ungroup()

# 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 <- 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)


# Record result
evaluation <- union(
  evaluation,
  data.frame(
    data = "model",
    model = "regularized linear: mu + b_movie + b_user",
    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: mu + b_movie + b_user 0.86581
```

2.3.3 Advanced Regularized Linear Model

In this step, the previously constructed model is expanded by incorporating additional predictors. To identify suitable predictors, the correlation of potential variables with the residual from the previous model is computed. Variables exhibiting the highest absolute correlation values are considered suitable for inclusion in the extended model.

```
# select possible predictor variables
predictors <- model_data_train  |>
  select( c( residual, !c(userId, movieId, rating, b_user, b_movie, rating_hat) ) )

# calculate all correlations with residual (absolute value)
corr <- abs(cor(predictors)[-1,1])
```



The previous model, initially based on two predictors b_{movie} and b_{user}, is now expanded into an extended regularized linear model. This advanced model incorporates the variables b_{movie}, b_{user}, b_{days} and b_{count}. Here, b_{days} captures the influence of the days since the first movie rating on the current rating, while b_{count} represents the impact of the count of ratings for the movie.

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

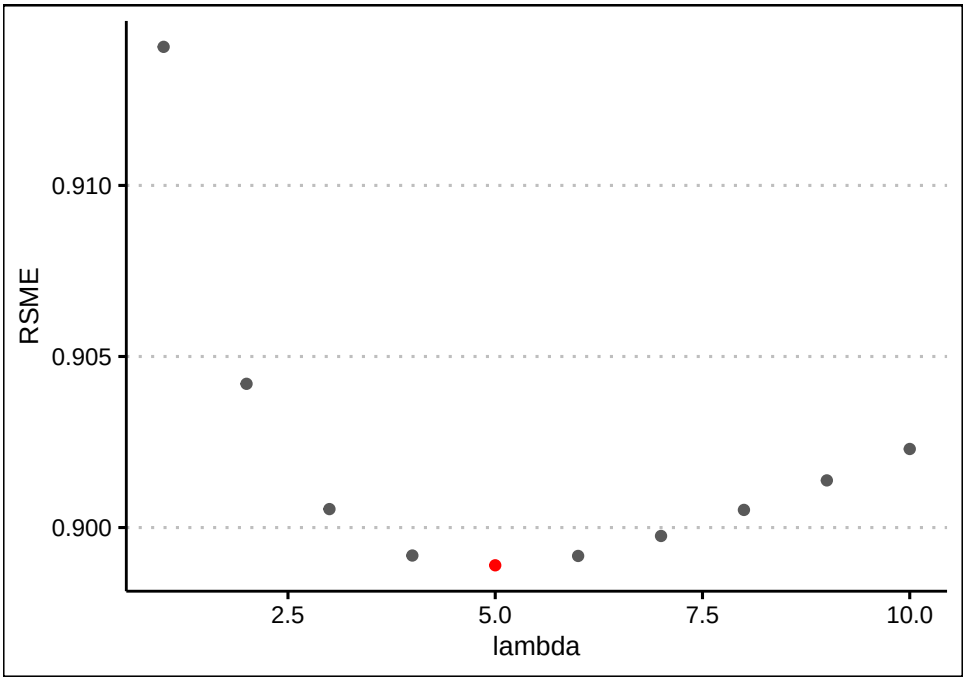
$$b_{\text{count}} = \frac{1}{n+\lambda}\sum_i (rating_i - \hat{\mu} - b_{\text{movie}} - b_{\text{user}} - b_{\text{days}}) = \sum_i \frac{rating_i - \hat{\mu} - b_{\text{movie}} - b_{\text{user}} - b_{\text{days}}}{n+\lambda}$$

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)

# use cross-validation to find lambda
train_advanced_linear_model <- function(lambda){

  # compute b_movie, b_user, b_days, and b_count
  small_data_train <- small_data_train |>
    group_by(movieId) |>
    mutate(b_movie = sum(rating - mu_hat) / (n() + lambda)) |>
    ungroup() |>
    group_by(userId) |>
```

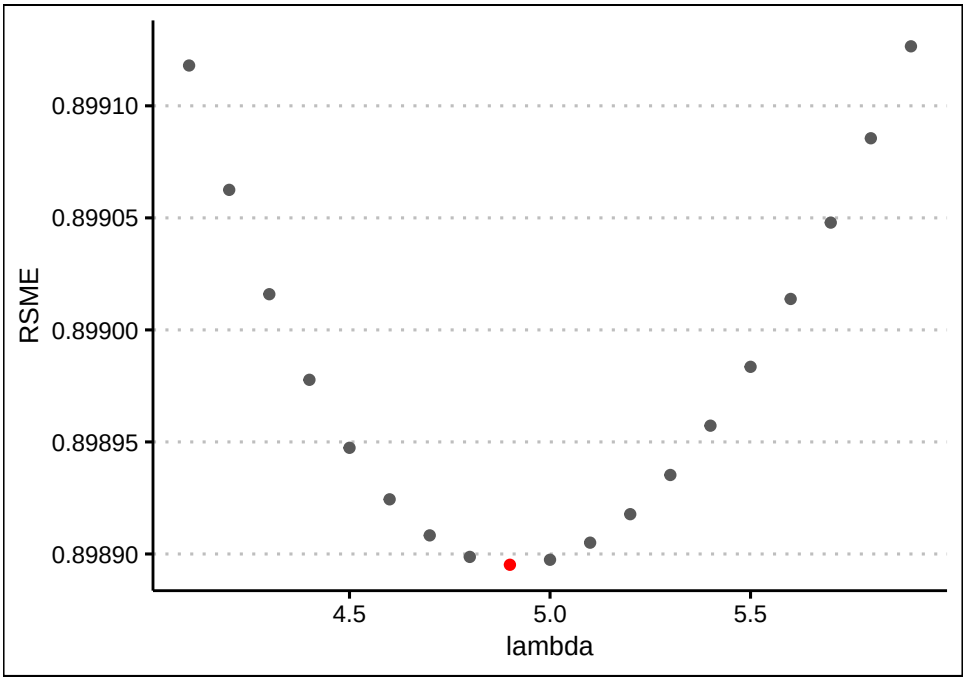
```
    mutate(b_user = sum(rating - mu_hat - b_movie) / (n() + lambda)) |>
    ungroup() |>
    group_by(days_since_movie_first_rating) |>
    mutate(b_days = sum(rating - mu_hat - b_movie - b_user) / (n() + lambda)) |>
    ungroup() |>
    group_by(movie_rating_count) |>
    mutate(b_count = sum(rating - mu_hat - b_movie - b_user - b_days) /
             (n() + lambda))

  # save b_movie, b_user, b_days, and b_count
  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))

  # compute prediction
  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)
}

# try 10 lambdas first for initial rough guess
lambdas <- seq(1, 10, 1)
rmses <- sapply(lambdas, train_advanced_linear_model)
lambda <- lambdas[which.min(rmses)]
print(lambda)
```

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




```
# try another 19 lambdas in the neighborhood of initial guess
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 the λ identified in the previous step, a regularized model featuring four predictor variables (*movie*, *user*, *days_since_movie_first_rating*, and *movie_rating_count*) is trained using the larger dataset *model_data*.

The residuals are once again saved for subsequent analysis and the resulting Root Mean Square Error (*RMSE*) is saved in *evaluation*.

```
# Calculate b_movie, b_user, b_days, and b_count
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()


# save b_movie, b_user, b_days, and b_count
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))

# Compute prediction
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)

# Record result
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: mu + b_movie + b_user 0.86581
## 3 model regularized linear: mu + b_movie + b_user + b_days + b_count 0.86498
```

2.3.4 Residual Models

While expanding the linear model with additional predictor variables is an option, it may lack novelty. The following approach explores alternative models like General Linear Model (GLM), Random Forest (RF), and Local Regression (loess) which were covered in the course. These models can more easily leverage all available predictor variables. However, attempts to compute these models using the functions in the *caret* library faced memory and CPU limitations.

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 for training a model
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, ]

  # train model for residual 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 GLM model, use small sample sizes to avoid too long run times
train_glm <- train_residual_model("glm", model_data_train, .05)

# 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

# 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: mu + b_movie + b_user 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 Ensemble Residual Model

In the final approach, two additional models, Random Forest (RF) and Local Regression (loess), for the residual are trained. Employing an ensemble strategy, all three models are amalgamated. The ensemble of the GLM, RF, and loess model is computed as a weighted average, with sample sizes serving as the weighting factors. This ensemble approach aims to harness the strengths of each model, fostering a more robust and comprehensive predictive framework.

```
# Two more models for ensemble
# ignore warnings from models
suppressWarnings({
  # train 2 more models, use different sample sizes to avoid too long run times
  train_rf <- train_residual_model("rf", model_data_train, .0001)
  train_loess <- train_residual_model("gamLoess", model_data_train, .002)

  # predict residuals with two more models
```

```
  predict_rf <- predict(train_rf, model_data_test)
  predict_loess <- predict(train_loess, model_data_test)

})
```

The ultimate prediction is calculated as the advanced linear model plus the weighted average of the three residual models. The resulting Root Mean Square Error (*RMSE*) is then stored in *evaluation*.

```
# weights derived from sample sizes
final_rating_hat <-
  rating_hat + (.05 * predict_glm + .0001 * predict_rf + .002 * predict_loess) /
               (.05 + .0001 + .002)

# Record result
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: mu + b_movie + b_user 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
```

This comprehensive approach, combining advanced linear modeling with the ensemble of residual models, provided the best predictive accuracy of all tested models.

3 Results

This chapter delves into the validation of the previously developed models, presenting the validation results. The models are now trained with the complete *edx* dataset and validated using the *final_holdout_test* dataset specifically set aside for this purpose. This validation process offers insights into the performance and predictive capabilities of the models on previously unseen data.

The process begins with the simple linear model, followed by the validation of the regularized simple model. Next, the more advanced linear model undergoes validation. Finally, the models that combine the advanced linear model with the residual models (GLM and ensemble) are validated.

In the case of regularized models, the λ parameter from the model construction phase is employed for validation.

3.1 Simple Linear Model

The simple linear model, defined by the formula

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

is constructed using the *edx* dataset and subsequently validated using the *final_holdout_test* dataset. Results are recorded in *evaluation*.

```
# build first model y_hat = mu_hat + b_movie + b_user + error
mu_hat <- mean(edx$rating)

# Movie and user effects
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()

# Test model, 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_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)

# Record result
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: mu + b_movie + b_user 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: mu + b_movie + b_user 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 Advanced Regularized Linear Model

Now, the advanced regularized linear model, incorporating the variables *days_since_movie_first_rating* and *movie_rating_count* in addition to *movie* and *user* is built using *edx* employing $\lambda = 4.9$. It is subsequently validated with *final_holdout_test*. The resulting *RSME* is saved in *evaluation*.

```
# Train model with days_since_movie_first_rating and movie_rating_count
# Use lambda from before
lambda <- 4.9
# compute b_movie, b_user, b_days, and b_count
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()


# Test model, save b_movie, b_user, b_days, and b_count
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))

# compute prediction
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)

# Record result
```

```r
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: mu + b_movie + b_user 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

Finally, three models for the residual, General Linear Model (GLM), Random Forest (RF), and Local Regression (loess), are trained on a small sample of *edx*. For validation, the prediction of the residual is added to the prediction from the advanced linear model. This process is executed initially with GLM and subsequently through an ensemble approach, combining GLM, RF, and loess. This method demonstrates the strengths of each residual model for an enhanced overall predictive outcome.

```r
# train GLM model with smaller sample size to avoid too long run times
train_glm <- train_residual_model("glm", edx, .05)

# predict residuals
predict_glm <- predict(train_glm, final_holdout_test)

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

# Record result
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: mu + b_movie + b_user 0.86581
## 3    model regularized linear: mu + b_movie + b_user + b_days + b_count 0.86498
```

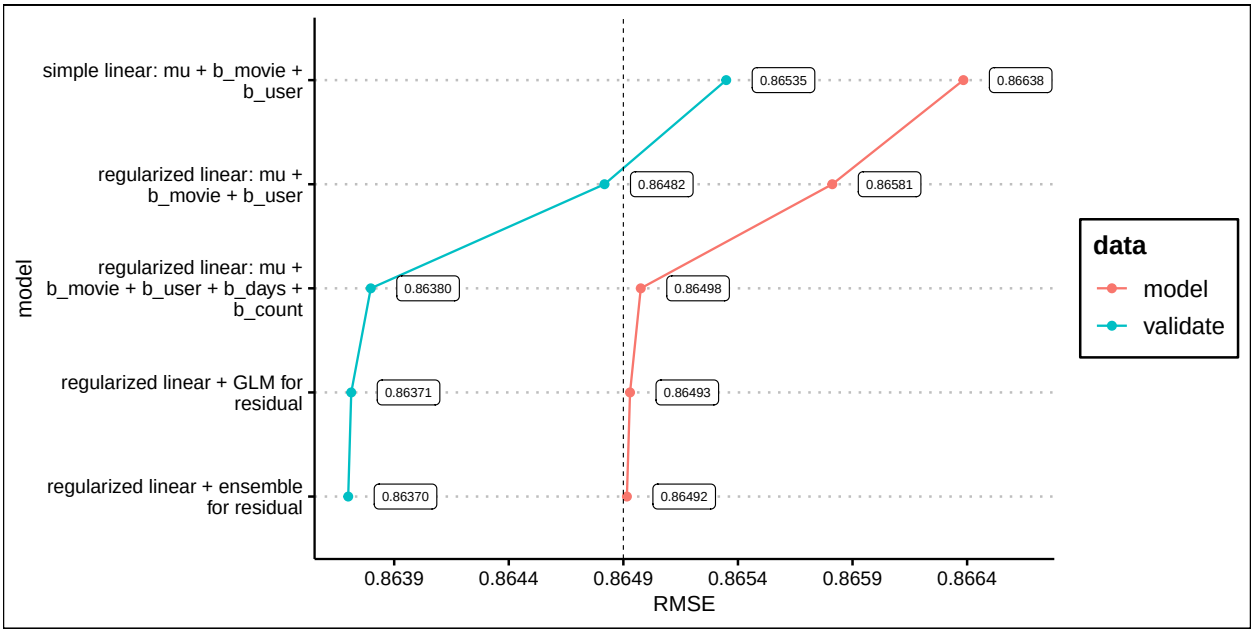
```
## 4    model                         regularized linear + GLM for residual 0.86493
## 5    model                    regularized linear + ensemble for residual 0.86492
## 6 validate                          simple linear: mu + b_movie + b_user 0.86535
## 7 validate                     regularized 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
```

```
# ignore warnings from models
suppressWarnings({

  # train 2 more models, use different sample sizes to avoid too long run times
  train_rf <- train_residual_model("rf", edx, .0001)
  train_loess <- train_residual_model("gamLoess", edx, .002)

    # predict residuals with two more models
  predict_rf <- predict(train_rf, final_holdout_test)
  predict_loess <- predict(train_loess, final_holdout_test)

})


# final prediction as advanced linear model + weighted average of three residual models
# weights derived from sample sizes
final_rating_hat <-
  rating_hat + (.05 * predict_glm + .0001 * predict_rf + .002 * predict_loess) /
               (.05 + .0001 + .002)

# Record result
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: mu + b_movie + b_user 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 Result Summary

The graph below provides an overview of the Root Mean Square Error (*RMSE*) for the various models during both the construction and validation phases. The most substantial improvements are observed with regularization and the addition of two more predictors to the linear model. While the residual models contribute to accuracy, their impact is comparatively smaller, possibly due to limited feasible sample sizes that may not fully capture data variability. Discrepancies in accuracy between model construction and validation may stem from utilizing the complete *edx* dataset for training in the validation phase.



4 Discussion

The final model showcases a fusion of various methodologies and successfully attains the predetermined Root Mean Square Error (RMSE) target for the course. Significant strides in precision are observed through the expansion of the initial simple linear model, which is based on user and movie bias, into a more intricate framework. This advanced model integrates additional features, such as the number of ratings a movie receives and the age of the movie at the time of rating.

As anticipated, the incorporation of regularization plays a major role in increasing the model's accuracy. This is evident not only in the context of the basic linear model but also extends to the more advanced linear model.

While the utilization of alternative models like General Linear Model (GLM), Random Forest (RF), and Local Regression (loess) faced constraints due to computational resources (CPU power and RAM), a pragmatic workaround involved training these models on a smaller subset of the data. This approach demonstrates the potential of these methods to effectively improve accuracy.

Other possible approaches were not further explored in this project. The influence of genres, particularly in the area of drama, suggests that the integration of genre variables into the extended model could further improve accuracy. However, this approach, although potentially beneficial, aligns closely with the existing methodology and lacks a significant element of novelty.

Moreover, the implementation of matrix factorization, leveraging the *recosystem* package, holds promise for superior results. However, this option was not pursued as the package was not part of the course.