HarvardX: PH125.9x Data Science
MovieLens Rating Prediction Project

Data Science: Capstone Project for Harvardx Professional Data Science Certificate
(Movielens Project: PH125.9x)

Delpagodage Ama Nayanahari Jayaweera

01 December 2024

Contents

1 Introduction

This project focuses on the development of a movie recommendation algorithm utilizing the publicly available MovieLens dataset. Inspired by the **Netflix Prize Challenge**, the objective is to design a recommendation system capable of accurately predicting the star ratings that users are likely to assign to specific movies on the Netflix platform.

The MovieLens dataset, obtained from the online repository of the GroupLens Research Lab, is divided into two subsets:

1. Edx Set: This subset will be used to develop and train the recommendation algorithm.
2. Validation Set: This subset will be employed to evaluate the performance of the algorithm at a later stage in the project.

A script provided by EdX (not included in this report) facilitates the downloading of the dataset and the creation of the two subsets. Below is a preview of the first few entries from the dataset.

2 Key points

1. This project presents models for building a movie recommendation system based on user ratings (graded out of 5).
2. The models account for the influence of four distinct variables on the ratings, incorporating regularization techniques to address small sample sizes.
3. Three versions of the model are provided, all achieving the target root mean squared error (RMSE) of 0.86490.

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

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

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

data_dir <- "D:/Edex/Final/Data_Science_Capstone_MovieLens/ml-10M100K"
ratings_file <- file.path(data_dir, "ratings.dat")
movies_file <- file.path(data_dir, "movies.dat")

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

if (!file.exists(ratings_file) || !file.exists(movies_file)) {
  stop("Required data files are missing. Please check file paths.")
}
```

```
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")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

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

```
head(edx,3)
```

```
  userId movieId rating timestamp           title                        genres
1      1     122      5 838985046 Boomerang (1992)                Comedy|Romance
2      1     185      5 838983525  Net, The (1995)         Action|Crime|Thriller
4      1     292      5 838983421  Outbreak (1995) Action|Drama|Sci-Fi|Thriller
```

```
head(validation,3)
```

```
  userId movieId rating timestamp               title
1      1     231      5 838983392 Dumb & Dumber (1994)
2      1     480      5 838983653 Jurassic Park (1993)
3      1     586      5 838984068    Home Alone (1990)
                          genres
1                         Comedy
2 Action|Adventure|Sci-Fi|Thriller
3                 Children|Comedy
```

The **edx** and **validation** subsets of the MovieLens dataset are structurally identical, ensuring consistency in data analysis and modeling. A brief descriptive analysis of the edx set is provided below to offer an overview of the dataset and outline the strategic approach for addressing the recommendation challenge.

The **edx** set contains 9000055 rows and 6 columns. As illustrated in the preview, the data is structured in a tidy format, with each entry representing a triplet, *[userId, movieId, rating]* contained in the first 3 columns.

This means each row corresponds to a rating given by a user for a specific movie.

- *timestamp*: the timecode at which the rating was given
- *title*: the title of the movie
- *genres*: all genres the movie belongs to, separated by colons

It is also insightful to note that the dataset is composed of 69878 unique users and 10677 unique movies. Storing this data in a user-movie matrix would result in an exceedingly large number of elements, as illustrated below.

```
n_distinct(edx$userId)*n_distinct(edx$movieId)
```

```
[1] 746087406
```

This number is about 80 times larger than the number of rows in the **edx** set, which means that the user-movie matrix would be very sparse.

```
round(nrow(edx)/(n_distinct(edx$userId)*n_distinct(edx$movieId))*100,1)
```

```
[1] 1.2
```

This implies that only 1.2% of the values in the matrix are non-missing (i.e., actual ratings), highlighting the significant sparsity of the data—most users have not rated most movies. The objective of this project is to address this sparsity by predicting the ratings users would assign to movies they have not yet rated.

To accomplish this, I will develop an algorithm based on the models introduced in the HarvardX Machine Learning course. The methodologies employed will be detailed in the **Analysis** section of this report.

To evaluate the performance of the algorithm, I will generate rating predictions for each user-movie pair within the **validation** dataset.

These predictions will be compared to the true ratings using the Root Mean Squared Error (RMSE) as defined by :

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

where:

- N is the number of user-movie combinations
- $y_{u,i}$ is the rating of movie i by user u
- $\hat{y}_{u,i}$ is the prediction

The RMSE target for this project is 0.86490. The prediction results and the final RMSE will be included in the **Results** section of this report.

2.1 Analysis

2.1.1 Data preparation

As mentioned in the introduction, the performance of the algorithm will be assessed using the RMSE - let's first define a RMSE function.

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

Then, the **validation** set is going to be used only for the final assessment of the algorithm in the **Results** section. To actually build the algorithm, it is relevant to split the **edx** set into a train and a test set. The following code achieves this.

```
# Test set will be 20% of edx dataset
set.seed(89, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

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

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

rm(test_index, temp, removed)
```

2.1.2 Baseline model

I will use the model based on movie and user bias proposed in the HarvardX Machine Learning course as a baseline. As a reminder, this model is defined by the equation:

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

where:

- $Y_{u,i}$ is the rating for user u watching movie i
- μ is the overall average rating
- b_i is the movie bias - the average rating for movie i
- b_u is the user bias - the average rating for user u
- $\varepsilon_{u,i}$ is an error term - the residual for user u movie i prediction

Let's build this model using the train set, make predictions on the test set and calculate the baseline RMSE.

```r
# Average of ratings in train set
mu <- mean(train_set$rating)

# Adding the movie effect b_i
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Adding the user effect b_u
b_u <- train_set %>%
  left_join(b_i, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Making predictions on test set
predictions <- test_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

baseline_rmse <- RMSE(predictions, test_set$rating)
print(paste("Baseline RMSE is",round(baseline_rmse,5)))
```

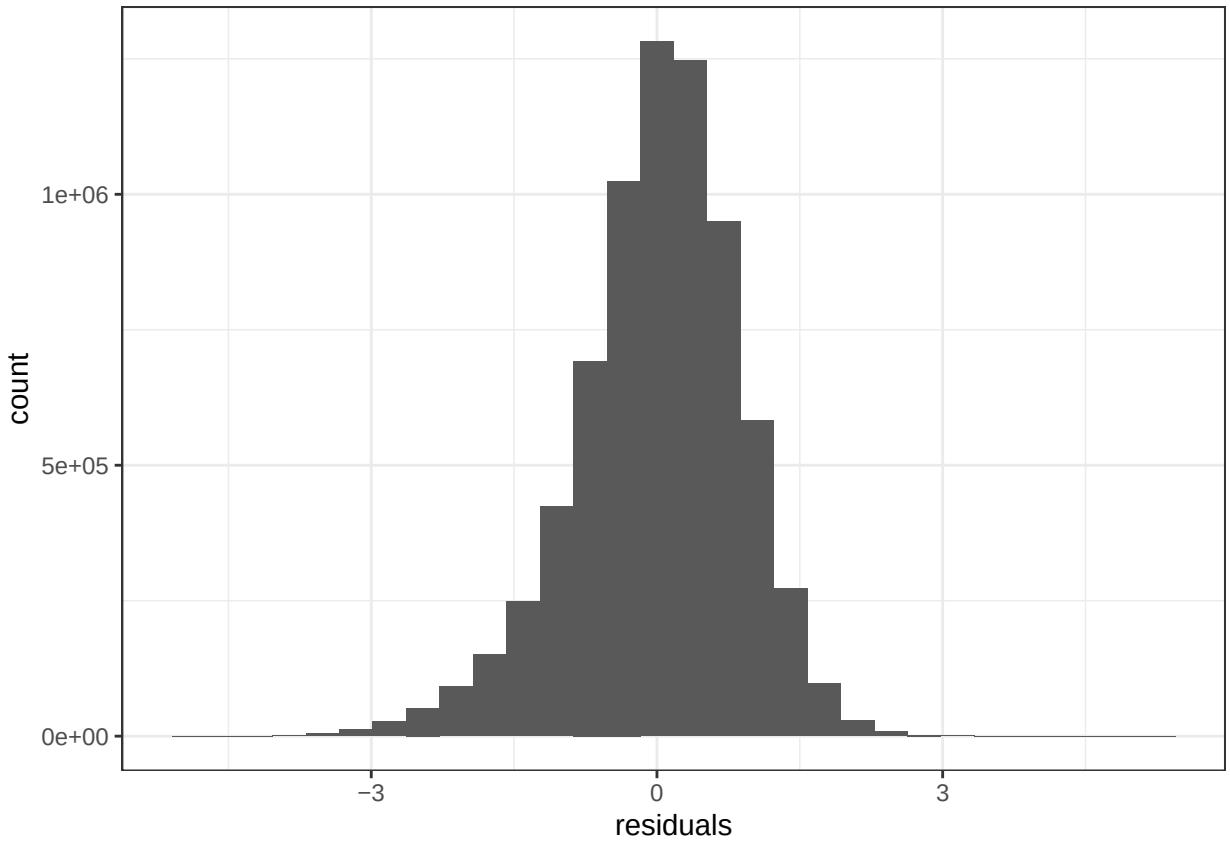
```
[1] "Baseline RMSE is 0.86724"
```

2.1.3 Exploratory and visual analysis

Based on the baseline model outlined above, I will conduct an initial exploration of the available variables in the training set. The goal is to identify strategies for improving the model and reducing the RMSE. To start, I will calculate the residuals of the baseline model and visualize them by plotting a histogram.

```r
# Add a residuals column to train set
train_set <- train_set %>%
  left_join(b_i, by = 'movieId') %>%
  left_join(b_u, by = 'userId') %>%
  mutate(residuals = rating - mu - b_i - b_u) %>%
  select(-b_i, -b_u)


# Plot a histogram of residuals
train_set %>%
  ggplot(aes(residuals)) + geom_histogram() + theme_bw()
```



We can see the residuals are still quite widely spread around 0, which means there is some room for improvement. I will now investigate which variables may have an effect on these residuals.

2.1.3.1 Additional effects First, I will use the *timestamp* column to determine the effect of rating date on the residuals. Let's transform the timecode contained in the column in a real date and round this date to the nearest month, then store the plot of the residuals vs. rating date.

```
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
library(lubridate)
train_set %>%
  mutate(date=round_date(as_datetime(timestamp),unit="month")) %>%
  group_by(date) %>%
  summarize(date=date[1], avg_res=mean(residuals), se_res= sd(residuals)/sqrt(n())) %>%
  ggplot(aes(date, avg_res, alpha=1/se_res)) +
  geom_point(show.legend = F) +
  ylim(c(-0.25, 0.75))+
  theme_bw() +
  theme(axis.title.y = element_blank()) -> plot_date
```

Then, I will use the *title* column to extract the movie release year as it is included in the title. The year is always into brackets so it is pretty easy to extract. Let's extract movie year and plot it against residuals, and store this plot

```
train_set %>%
  mutate(year=as.numeric(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
```

```
  group_by(year) %>%
  summarize(year=year[1], avg_res=mean(residuals), se_res = sd(residuals)/sqrt(n())) %>%
  ggplot(aes(year, avg_res, alpha=1/se_res))+
  geom_point(show.legend = F)+
  ylim(c(-0.25, 0.75))+
  theme_bw()+
   theme(axis.title.y = element_blank()) -> plot_year
```

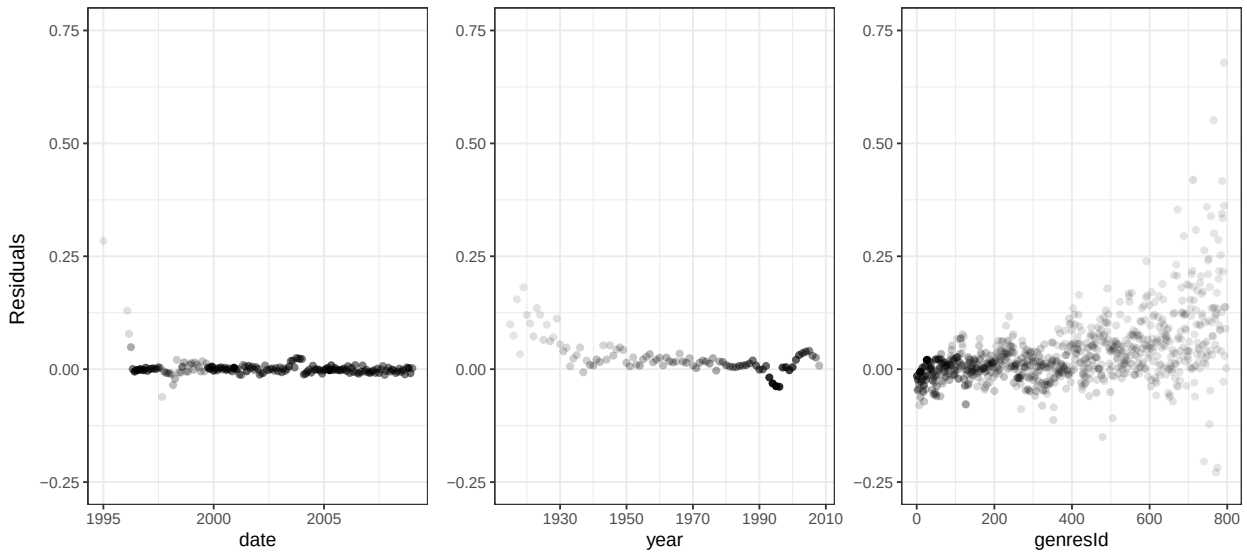
Finally, I will use the *genres* column to extract the movie genres. To simplify the analysis, I will treat each unique combination of genres as a distinct genre. I will then stratify the data by genre and plot the residuals against these genres. For better readability, instead of displaying long genre combinations on the x-axis, I will assign an arbitrary number, genreId, to each genre combination. This will allow for a more concise and interpretable plot.

```
genres_list <- unique(train_set$genres)

train_set %>%
  group_by(genres) %>%
  summarize(genres=genres[1], n_ratings=n(), avg_res=mean(residuals),
            se_res=ifelse(n()>1,sd(residuals)/sqrt(n()),1000)) %>%
  mutate(genresId=match(genres,genres_list)) %>%
  ggplot(aes(genresId, avg_res, alpha=1/se_res))+
  geom_point(show.legend = F)+
  ylim(c(-0.25, 0.75))+
  theme_bw() +
  theme(axis.title.y = element_blank()) -> plot_genres
```

To facilitate comparison, I will plot the three effects side by side. In each scatterplot, the opacity of each data point will be adjusted according to the standard error: the less opaque the data point, the higher the standard error, indicating greater uncertainty in the residual. This will provide a clearer visual representation of the residuals and their associated uncertainty across the different effects.

```
# plots next to each other
library(gridExtra)
grid.arrange(plot_date, plot_year, plot_genres, nrow=1, left="Residuals")
```



Upon analyzing the effects, it appears that the rating date does not significantly influence the residuals. The movie release year shows a moderate impact, particularly for older films (with a high standard error) and those released after 1990. However, the most notable effect comes from the genres, which seem to have the strongest influence on the residuals, despite occasional high standard errors. This suggests that genre-related factors might be contributing significantly to the model's performance, even if the data uncertainty is relatively high in some cases.

Thus, to improve RMSE I propose to add a genre and a release year effect to the baseline model. With g_i the genre and r_i the release year for movie i, I will try to fit the following model:

$$Y_{u,i} = \mu + b_i + b_u + b_k + b_n + \varepsilon_{u,i} \text{ with } k = g_i \text{ and } n = r_i$$

Let's build this model using the train set, make predictions on the test set and calculate the first model RMSE.

```
# Clearing the predictions from old models
rm(predictions)

# Adding the genres effect b_k
b_k <- train_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  group_by(genres) %>%
  summarize(b_k = mean(rating - mu - b_i - b_u))

# Adding the release year effect b_n
b_n <- train_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_k, by='genres') %>%
  mutate(year=as.factor(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
  group_by(year) %>%
  summarize(b_n = mean(rating - mu - b_i - b_u - b_k))

# Making predictions on test set
predictions <- test_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_k, by='genres') %>%
  mutate(year=as.factor(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
  left_join(b_n, by='year') %>%
  mutate(pred = mu + b_i + b_u + b_k + b_n) %>%
  pull(pred)

mymodel_1_rmse <- RMSE(predictions, test_set$rating)
print(paste("RMSE with my model #1 is",round(mymodel_1_rmse,5)))
```

```
[1] "RMSE with my model #1 is 0.86669"
```

The RMSE is slightly better than baseline. To do even better I will now use regularization techniques.

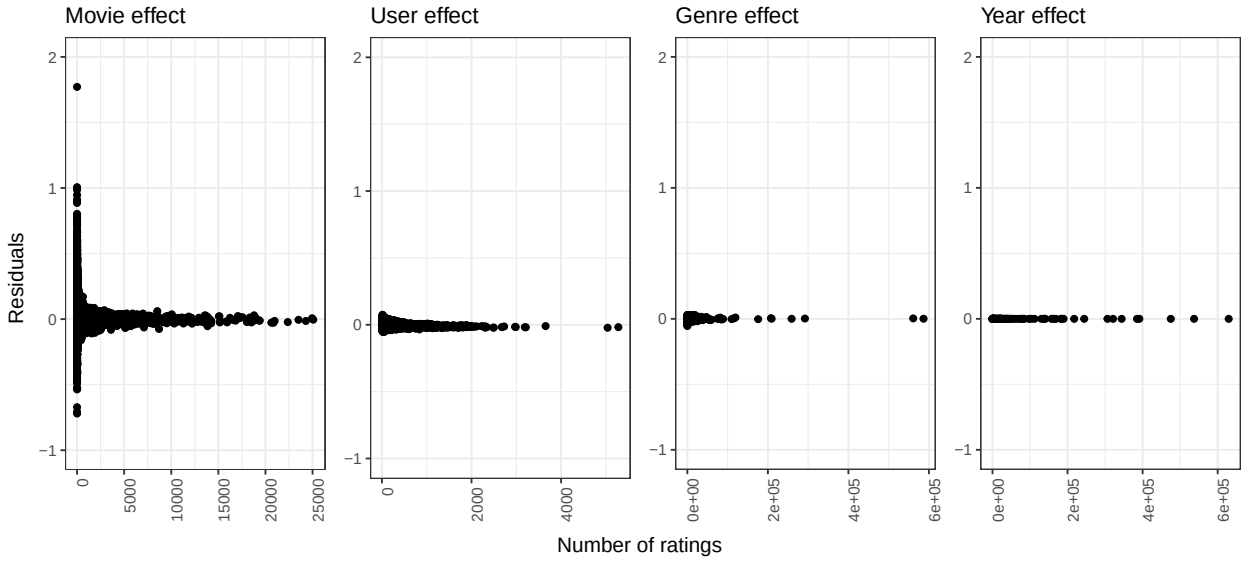
2.1.3.2 Number of ratings - regularization

Let's update the residuals of the new model so we can explore the influence of the number of ratings.

```r
# Update the residuals column in train set
train_set <- train_set %>%
  left_join(b_i, by = 'movieId') %>%
  left_join(b_u, by = 'userId') %>%
  left_join(b_k, by = 'genres') %>%
  mutate(year=as.factor(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
  left_join(b_n, by = 'year') %>%
  mutate(residuals = rating - mu - b_i - b_u - b_k - b_n) %>%
  select(-b_i, -b_u, -b_k, -b_n)
```

Now, it will plot the residuals against the number of ratings, stratified by movie, user, genres, and release year. The approach to creating these plots will follow a similar process to the previous residuals plot, though the specific code for generating these plots is not shown here. These visualizations will help to further explore the influence of these factors on the residuals and provide deeper insights into model performance.

```r
# plots next to each other
library(gridExtra)
grid.arrange(plot_movie, plot_user, plot_genres, plot_year, nrow=1,
             left="Residuals", bottom="Number of ratings")
```



It seems that the residuals are indeed higher for movies with fewer ratings, which is particularly noticeable for the movie effect. This outcome is expected, as movies with a low number of ratings have a higher standard error in estimating their average rating (or the average rating for a given user, genre, or release year). This increased uncertainty in the estimation leads to larger residuals, reflecting the model's difficulty in making accurate predictions for these less-rated movies. There are even 202 movies (i.e. 1.9 % of the movies) for which only 1 rating is available, which leads obviously to overtraining my model.

To account for these low number of ratings I am going to penalize the estimates coming from small sample sizes. If b_x is one of the effects included in my first model, I am now going to calculate b_x not as the average rating for a given x, but as:

$$b_x(\lambda) = \frac{1}{\lambda + n_x} \sum_{x=1}^{n_x} (Y_{u,i} - \mu)$$

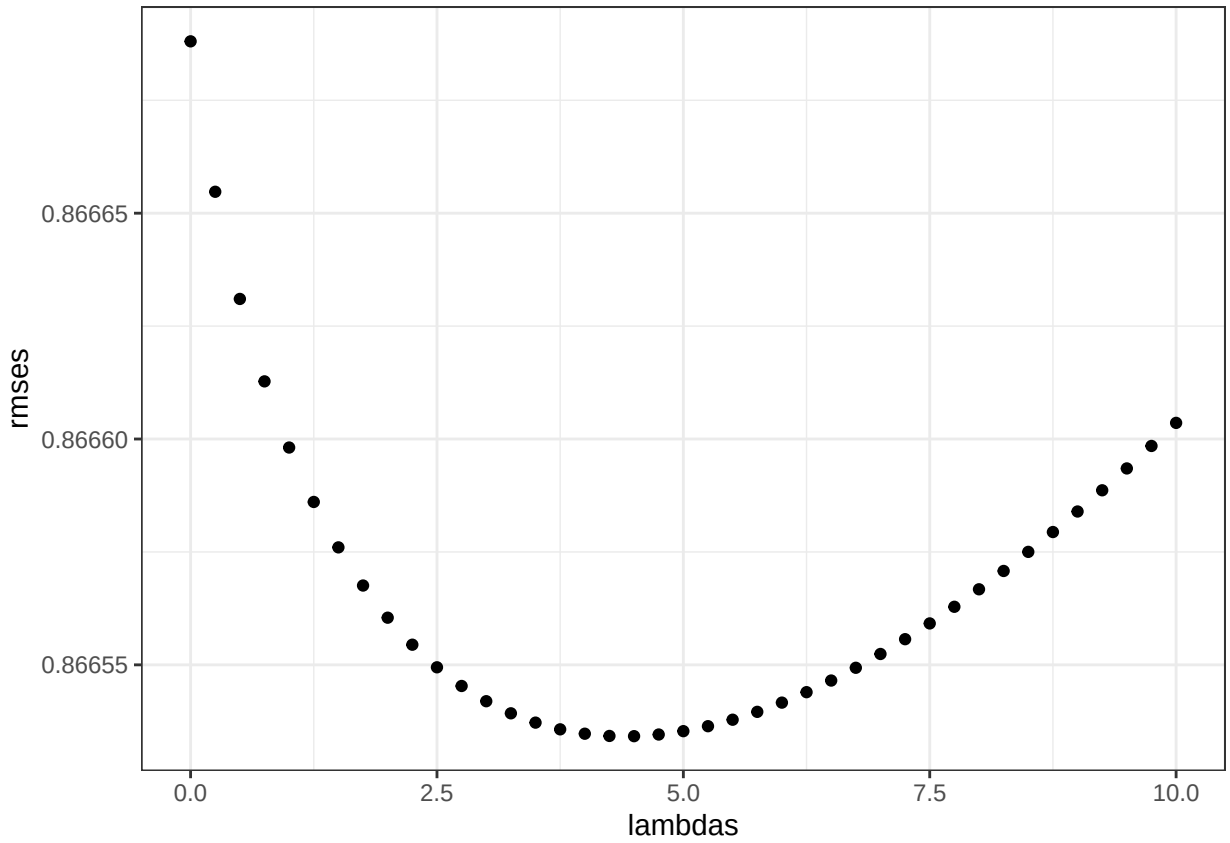
λ is a tuning parameter and I am going to use cross-validation to determine the value of λ that minimizes the RMSE.

So let's first regularize movie effect only. I will apply a range of lambdas on the calculation of b_i and plot the obtained RMSEs.

```
# Regularization for movie effect
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  b_i_reg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i_reg = sum(rating - mu)/(n()+l))
  b_u_2 <- train_set %>%
    left_join(b_i_reg, by = 'movieId') %>%
    group_by(userId) %>%
    summarize(b_u_2 = mean(rating - b_i_reg - mu))
  b_k_2 <- train_set %>%
    left_join(b_i_reg, by = 'movieId') %>%
    left_join(b_u_2, by='userId') %>%
    group_by(genres) %>%
    summarize(b_k_2 = mean(rating - b_i_reg - b_u_2 - mu))
  b_n_2 <- train_set %>%
    left_join(b_i_reg, by = 'movieId') %>%
    left_join(b_u_2, by='userId') %>%
    left_join(b_k_2, by='genres') %>%
    group_by(year) %>%
    summarize(b_n_2 = mean(rating - b_i_reg - b_u_2 - b_k_2 - mu))

  predicted_ratings <-
    test_set %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_2, by='userId') %>%
    left_join(b_k_2, by='genres') %>%
    mutate(year=as.factor(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
    left_join(b_n_2, by='year') %>%
    mutate(pred = mu + b_i_reg + b_u_2 + b_k_2 + b_n_2) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

tibble(lambdas=lambdas, rmses=rmses) %>%
  ggplot(aes(lambdas, rmses)) +
  geom_point()+
  theme_bw()
```



We can then extract the value of λ that minimizes the RMSE and the minimum RMSE for this second model.

```
l_i <- lambdas[which.min(rmses)]
mymodel_2_rmse <- min(rmses)
print(paste("Optimal lambda is", l_i))
```

```
[1] "Optimal lambda is 4.5"
```

```
print(paste("RMSE with my model #2 is",round(mymodel_2_rmse,5)))
```

```
[1] "RMSE with my model #2 is 0.86653"
```

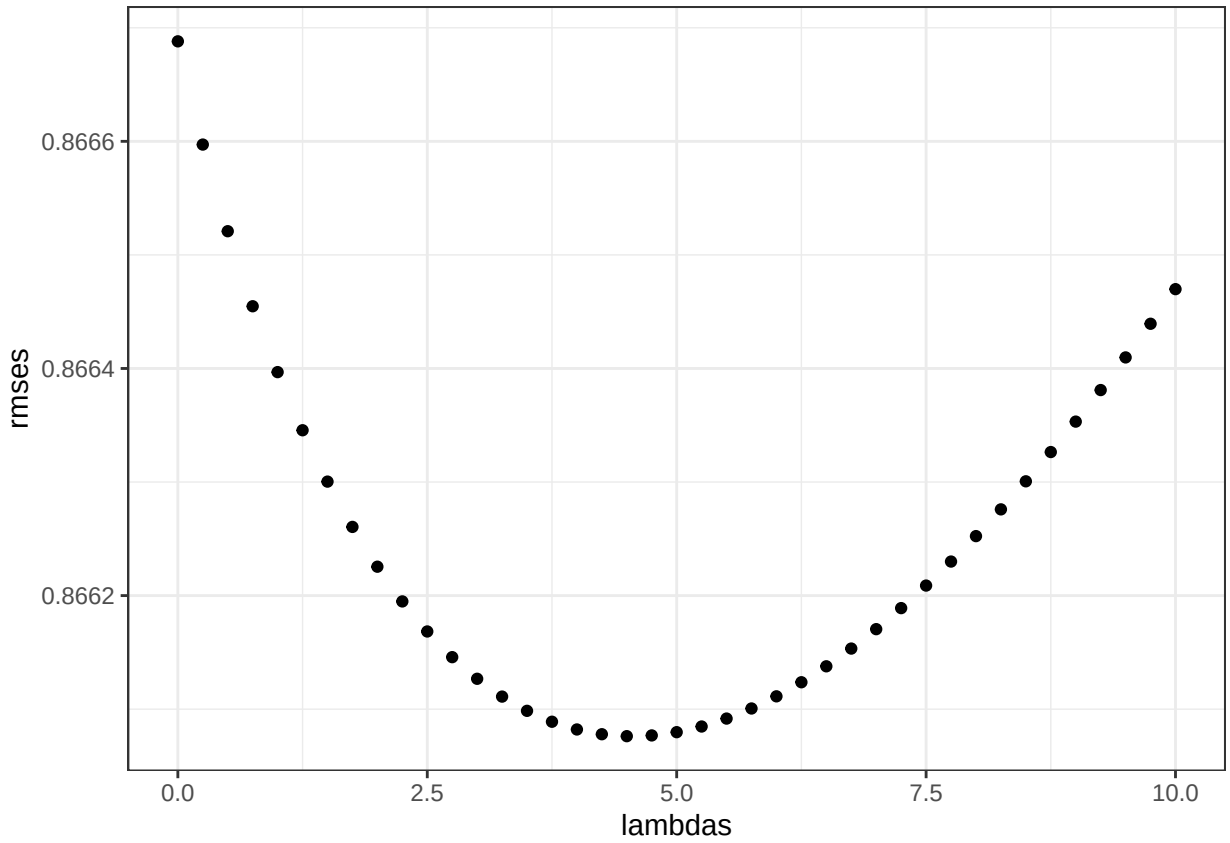
The RMSE has shown some improvement compared to model 1. To further enhance the model's performance, I will implement regularization on all the effects in the model. This will help control overfitting, especially for factors with high variance or less data, and ultimately lead to more reliable predictions and a better RMSE. Let's apply a set of λ on the calculation of b_i, b_u, b_k and b_n and plot the obtained RMSEs.

```
# Regularization for all effects
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  b_i_reg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i_reg = sum(rating - mu)/(n()+l))
```

```
  b_u_reg <- train_set %>%
    left_join(b_i_reg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_reg = sum(rating - b_i_reg - mu)/(n()+l))
  b_k_reg <- train_set %>%
    left_join(b_i_reg, by="movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_k_reg = sum(rating - b_i_reg - b_u_reg - mu)/(n()+l))
  b_n_reg <- train_set %>%
    left_join(b_i_reg, by="movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    left_join(b_k_reg, by="genres") %>%
    mutate(year=as.factor(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
    group_by(year) %>%
    summarize(b_n_reg = sum(rating - b_i_reg - b_u_reg - b_k_reg - mu)/(n()+l))

  predicted_ratings <-
    test_set %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    left_join(b_k_reg, by='genres') %>%
    mutate(year=as.factor(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
    left_join(b_n_reg, by='year') %>%
    mutate(pred = mu + b_i_reg + b_u_reg + b_k_reg + b_n_reg) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

tibble(lambdas=lambdas, rmses=rmses) %>%
  ggplot(aes(lambdas, rmses)) +
  geom_point()+
  theme_bw()
```



We then extract the value of λ that minimizes the RMSE and the minimum RMSE for this third model.

```
lambda <- lambdas[which.min(rmses)]
mymodel_3_rmse <- min(rmses)
print(paste("Optimal lambda is", lambda))
```

```
[1] "Optimal lambda is 4.5"
```

```
print(paste("RMSE with my model #3 is",round(mymodel_3_rmse,5)))
```

```
[1] "RMSE with my model #3 is 0.86608"
```

The RMSE does not further improve and we observe that our regularization parameter λ is the same if we regularize movie effect only or all effects. This approach is logical, as we observed earlier that the movie effect was significantly influenced by the low number of ratings for some movies. By regularizing the effects, we aim to mitigate this issue and improve the model's generalization. Based on this analysis, I have designed three models that I will now apply to the validation set. This will allow me to assess their final performance and determine which model provides the best results.

2.2 Results

Let's apply all the models I studied in the **Analysis** section and compare them in a table.

```
rm(predictions) # Clearing predictions from old model

# Baseline model - append _f suffix for 'final'
mu_f <- mean(edx$rating)

b_i_f <- edx %>%
  group_by(movieId) %>%
  summarize(b_i_f = mean(rating - mu_f))

b_u_f <- edx %>%
  left_join(b_i_f, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u_f = mean(rating - mu_f - b_i_f))

predictions_baseline <- validation %>%
  left_join(b_i_f, by='movieId') %>%
  left_join(b_u_f, by='userId') %>%
  mutate(pred = mu_f + b_i_f + b_u_f) %>%
  pull(pred)

baseline_rmse_f <- RMSE(predictions_baseline, validation$rating)


# Model #1 - append _f suffix for 'final'
b_k_f <- edx %>%
  left_join(b_i_f, by='movieId') %>%
  left_join(b_u_f, by='userId') %>%
  group_by(genres) %>%
  summarize(b_k_f = mean(rating - mu_f - b_i_f - b_u_f))

b_n_f <- edx %>%
  left_join(b_i_f, by='movieId') %>%
  left_join(b_u_f, by='userId') %>%
  left_join(b_k_f, by='genres') %>%
  mutate(year=as.factor(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
  group_by(year) %>%
  summarize(b_n_f = mean(rating - mu_f - b_i_f - b_u_f - b_k_f))

predictions_model1 <- validation %>%
  left_join(b_i_f, by='movieId') %>%
  left_join(b_u_f, by='userId') %>%
  left_join(b_k_f, by='genres') %>%
  mutate(year=as.factor(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
  left_join(b_n_f, by='year') %>%
  mutate(pred = mu_f + b_i_f + b_u_f + b_k_f + b_n_f) %>%
  pull(pred)

mymodel_1_rmse_f <- RMSE(predictions_model1, validation$rating)


# Model #2 - append _f2 suffix for 'final #2'
b_i_f2 <- edx %>%
    group_by(movieId) %>%
```

```
    summarize(b_i_f2 = sum(rating - mu_f)/(n()+l_i))
b_u_f2 <- edx %>%
    left_join(b_i_f2, by = 'movieId') %>%
    group_by(userId) %>%
    summarize(b_u_f2 = mean(rating - b_i_f2 - mu_f))
b_k_f2 <- edx %>%
    left_join(b_i_f2, by = 'movieId') %>%
    left_join(b_u_f2, by='userId') %>%
    group_by(genres) %>%
    summarize(b_k_f2 = mean(rating - b_i_f2 - b_u_f2 - mu_f))
b_n_f2 <- edx %>%
    left_join(b_i_f2, by = 'movieId') %>%
    left_join(b_u_f2, by='userId') %>%
    left_join(b_k_f2, by='genres') %>%
    mutate(year=as.factor(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
    group_by(year) %>%
    summarize(b_n_f2 = mean(rating - b_i_f2 - b_u_f2 - b_k_f2 - mu_f))

predictions_model2 <- validation %>%
  left_join(b_i_f2, by='movieId') %>%
  left_join(b_u_f2, by='userId') %>%
  left_join(b_k_f2, by='genres') %>%
  mutate(year=as.factor(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
  left_join(b_n_f2, by='year') %>%
  mutate(pred = mu_f + b_i_f2 + b_u_f2 + b_k_f2 + b_n_f2) %>%
  pull(pred)

mymodel_2_rmse_f <- RMSE(predictions_model2, validation$rating)

# Model #3 - append _f3 suffix for 'final #3'
b_i_f3 <- edx %>%
    group_by(movieId) %>%
    summarize(b_i_f3 = sum(rating - mu_f)/(n()+lambda))
b_u_f3 <- edx %>%
    left_join(b_i_f3, by = 'movieId') %>%
    group_by(userId) %>%
    summarize(b_u_f3 = sum(rating - b_i_f3 - mu_f)/(n()+lambda))
b_k_f3 <- edx %>%
    left_join(b_i_f3, by = 'movieId') %>%
    left_join(b_u_f3, by='userId') %>%
    group_by(genres) %>%
    summarize(b_k_f3 = sum(rating - b_i_f3 - b_u_f3 - mu_f)/(n()+lambda))
b_n_f3 <- edx %>%
    left_join(b_i_f3, by = 'movieId') %>%
    left_join(b_u_f3, by='userId') %>%
    left_join(b_k_f3, by='genres') %>%
    mutate(year=as.factor(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
    group_by(year) %>%
    summarize(b_n_f3 = sum(rating - b_i_f3 - b_u_f3 - b_k_f3 - mu_f)/(n()+lambda))

predictions_model3 <- validation %>%
  left_join(b_i_f3, by='movieId') %>%
  left_join(b_u_f3, by='userId') %>%
```

```
  left_join(b_k_f3, by='genres') %>%
  mutate(year=as.factor(str_extract(title,"(?<=\\()\\d{4}(?=\\))"))) %>%
  left_join(b_n_f3, by='year') %>%
  mutate(pred = mu_f + b_i_f3 + b_u_f3 + b_k_f3 + b_n_f3) %>%
  pull(pred)

mymodel_3_rmse_f <- RMSE(predictions_model3, validation$rating)



rmse_results <- tibble(model = c("Baseline", "Model 1", "Model 2", "Model 3"),
                       method=c("Movie + user effects",
                                "Movie + user + genre + year effects",
                                "Movie effect regularization",
                                "All effects regularization"),
                           RMSE = c(round(baseline_rmse_f,5),
                                    round(mymodel_1_rmse_f,5),
                                    round(mymodel_2_rmse_f,5),
                                    round(mymodel_3_rmse_f,5)))
rmse_results %>% knitr::kable()
```

model	method	RMSE
Baseline	Movie + user effects	0.86535
Model 1	Movie + user + genre + year effects	0.86476
Model 2	Movie effect regularization	0.86461
Model 3	All effects regularization	0.86429

We can see the lowest achieved RMSE is 0.86429 with Model 3 corresponding to All effects regularization. The target RMSE of 0.86490 is met for all 3 models.

Let's make 2 discussion comments about these results.

1. **Model Performance:** The most performing model I designed shows an RMSE improvement of only 0.12 % compared to the baseline model. This suggests that the movie and user effects already capture a significant portion of the variability in ratings. The genre and year effects, which were added to improve the model, provided only marginal improvements. This outcome makes sense, as both genre and year are intrinsic features of the movie, making them somewhat redundant with the movie effect itself.
2. **RMSE Analysis:** The lowest achieved RMSE of 0.86429 remains relatively high. This indicates that, on average, predictions for a given user-movie combination would be quite off. For instance, if I were to predict a rating of 4.3 for a movie-user pair that actually has a rating of 3.5, the model's prediction would still have a considerable error. This highlights the difficulty of accurately predicting ratings and the inherent challenges in the rating prediction task.

One limitation of this piece of work can be that I used only a one-fold cross-validation to tune the regularization parameters. It may have been interesting to use multiple-folds cross-validation to obtained better tuned λs.

Also, to further improve the model, I thought of collaborative filtering techniques to try to build user similarities. Basically, I would calculate a movie effect not using the average of all ratings but of the ratings

given by users that are *similar* to the user I want to give a prediction to. I tried to work on such models from the present dataset but I crashed R a few times because of the high computation demands.

I did not want to use prepackaged libraries like *recommenderlab* because I wanted to understand and code myself a collaborative filtering algorithm. A way to adapt to computational demands could be to use sparse matrices.

2.3 Conclusion

This project aimed at providing movie recommendations based on a dataset of previously known user-movie ratings. More specifically, the idea was to predict the rating a given user would give to a movie she/he had not seen already.

The general approach was to design a model based on the effect of different variables on user-movie rating. I included 4 effects in this model: movie, user, movie genre and movie release year. I also regularized these effects to account for the low number of ratings observed for some movies/users/genres/years.

The best result obtained on the **validation** set is a RMSE of 0.86429 which meets the target of 0.86490. I would think of collaborative filtering as a future approach to build on the present model and further improve the RMSE.