HarvardX PH125.9x - Data Science: Capstone

Sebastien Pavailler

14/11/2022

Key points

- This project provides models to build a movie recommendation system based on user-movie ratings (grade out of 5).
- The models include the effects of 4 different variables on rating with regularization for small sample sizes.
- Three model versions are presented that all meet the root mean squared error target of 0.86490.

I. Introduction

The project consists in creating a movie recommendation algorithm using publicly available data: the MovieLens dataset. This dataset is inspired from the so-called *Netflix challenge* in which the objective was to build a recommendation system to predict how many stars a given user of the Netflix service would give a specific movie. The MovieLens dataset is imported from the online repository of the GroupLens research lab, and split into two subsets: the **edx** set that will be used in the following computations to develop the recommendation algorithm, and the **validation** set that will be used once at the end of the project to test the performance of the developped algorithm. A piece of code provided by EdX (not shown in the report) allows to download the data and generate the two sets - a preview of the first entries are shown below.

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

```
# A tibble: 3 x 6
  userId movieId rating timestamp title            genres
   <dbl>   <dbl>  <dbl>     <dbl> <chr>            <chr>
1      1     122      5 838985046 Boomerang (1992) Comedy|Romance
2      1     185      5 838983525 Net, The (1995)  Action|Crime|Thriller
3      1     292      5 838983421 Outbreak (1995)  Action|Drama|Sci-Fi|Thriller
```

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

```
# A tibble: 3 x 6
  userId movieId rating timestamp title                genres
   <dbl>   <dbl>  <dbl>     <dbl> <chr>                <chr>
1      1     231      5 838983392 Dumb & Dumber (1994) Comedy
2      1     480      5 838983653 Jurassic Park (1993) Action|Adventure|Sci-Fi|~
3      1     586      5 838984068 Home Alone (1990)    Children|Comedy
```

As the **edx** and **validation** set are two different subsets of the MovieLens dataset, we can see they have the exact same structure. A brief descriptive analysis of the **edx** set is provided below in order to present the data and explain the general approach to address the challenge.

The **edx** set has 9000055 rows and 6 columns. As shown in the preview above, the data is organized in a tidy format where each entry corresponds to a triplet [userId, movieId, rating] contained in the first 3 columns. Thus, each entry is one rating given by one user to one movie. The other 3 columns are:

- *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. Thus, if one would like to store the data in a user-movie matrix, the number of elements would be huge (see 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 means that only 1.2 % of the values in the matrix would *not* be NAs - in other words, not every user rated every movie, far from it.

> The objective of the project is to reduce the sparsity of this matrix by predicting the rating users would give to movies they have not seen already.

To achieve this, I will develop an algorithm that builds up on the models proposed in the HarvardX Machine Learning course. The methods employed will be described in the **Analysis** section of this report. In order to assess the performance of the algorithm, I will produce rating predictions for each user-movie combination in the **validation** set. 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.

II. Analysis

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. 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)))
```

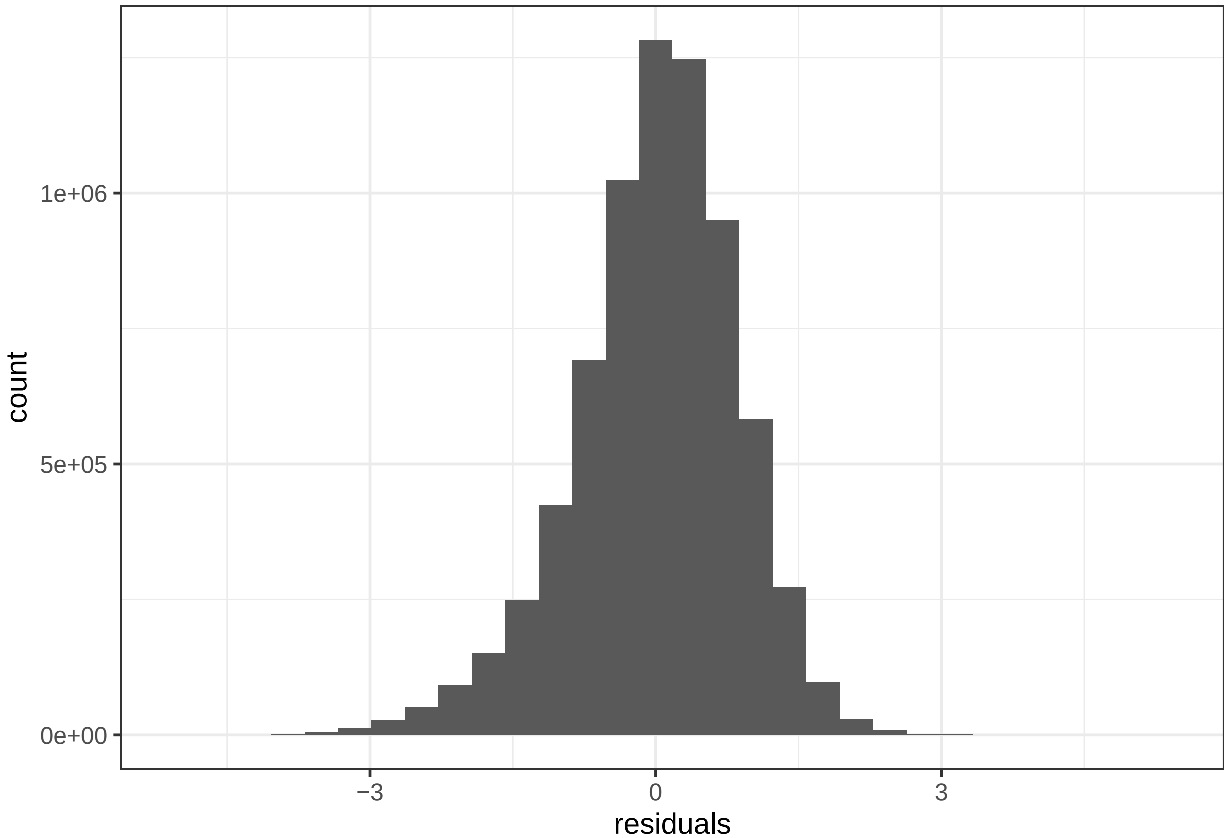
```
[1] "Baseline RMSE is 0.86724"
```

3. Exploratory and visual analysis

From the baseline model described above, I will perform a quick data exploration on the available variables in the train set in order to define the ways to improve the model and decrease the RMSE. Let's first calculate the residuals of the baseline model and plot them as 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.

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.

```
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
```

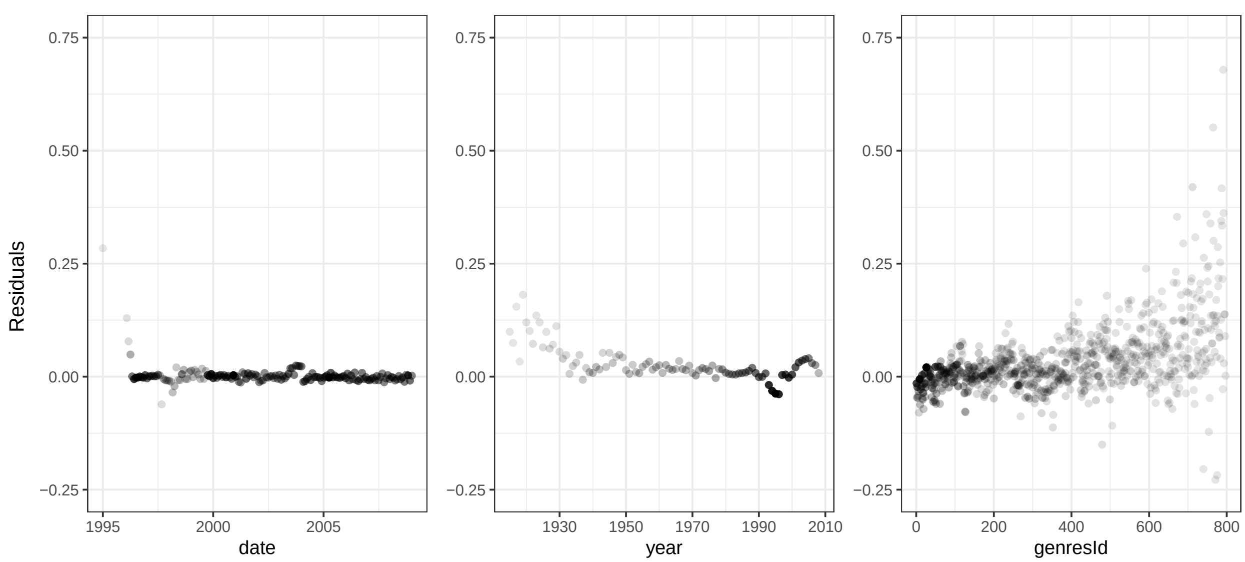
Finally, I will use the *genres* column to extract the movie genres. A specific movie may belong to many different genres, but to keep it simple I will consider any genres combination as a unique genre. Let's stratify by genre and store the plot against residuals. To make the plot easier to read I will not include the genre name on the x-axis (which can be a quite long combination of different genres), but rather an arbitrary number that I will call *genreId*.

```
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
```

Let's plot the 3 effects next to each other to be able to compare them. The opacity of each datapoint in the scatterplots is tuned to the standard error i.e. the less opaque the higher the standard error so the more uncertain the residual.

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



It appears that rating date does not seem to have that much of an effect on the residuals. Movie release year seems to have a moderate effect, especially for old movies (but the standard error is quite high) and for movies released after 1990. Genres seems to have the highest effect even though standard errors are sometimes quite high.

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"
```

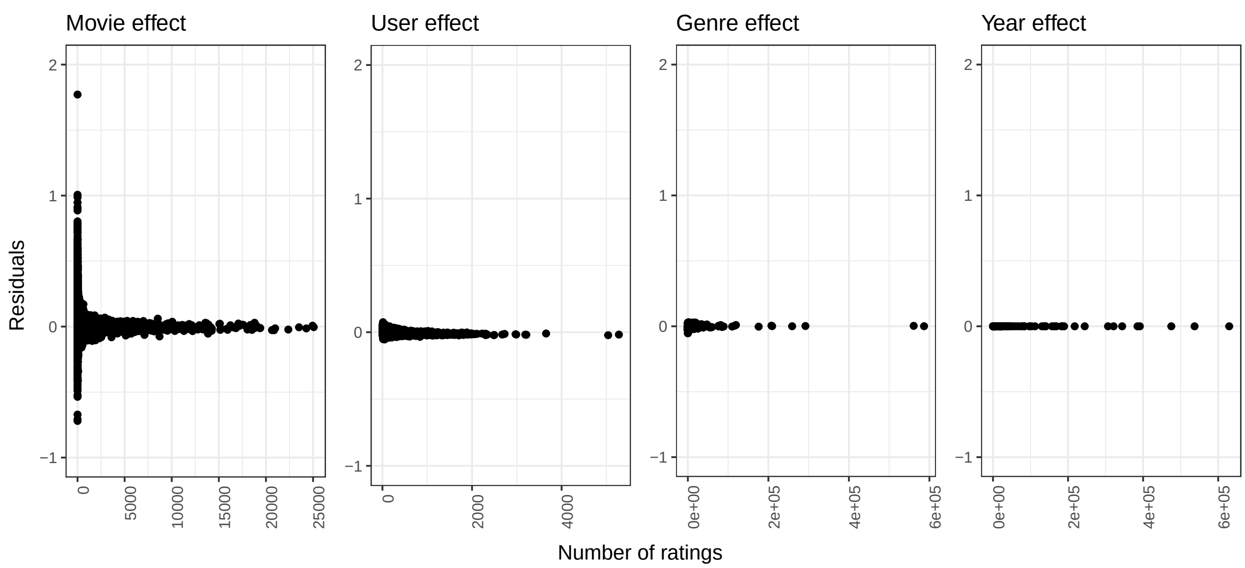
The RMSE is slightly better than baseline. To do even better I will now use regularization techniques.

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.

```
# 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 I can plot the residuals against the number of ratings stratified by movie, user, genres and release year. The code to create the 4 plots is not shown here but is similar to the previous residuals plot creation.

```
# 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 appears that the residuals are higher for lower number of ratings - this is especially true for the movie effect. This is expected as for a low number of ratings, the standard error of the estimate (the average rating for a given movie/user/genre/year) is expected to be higher. 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:

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

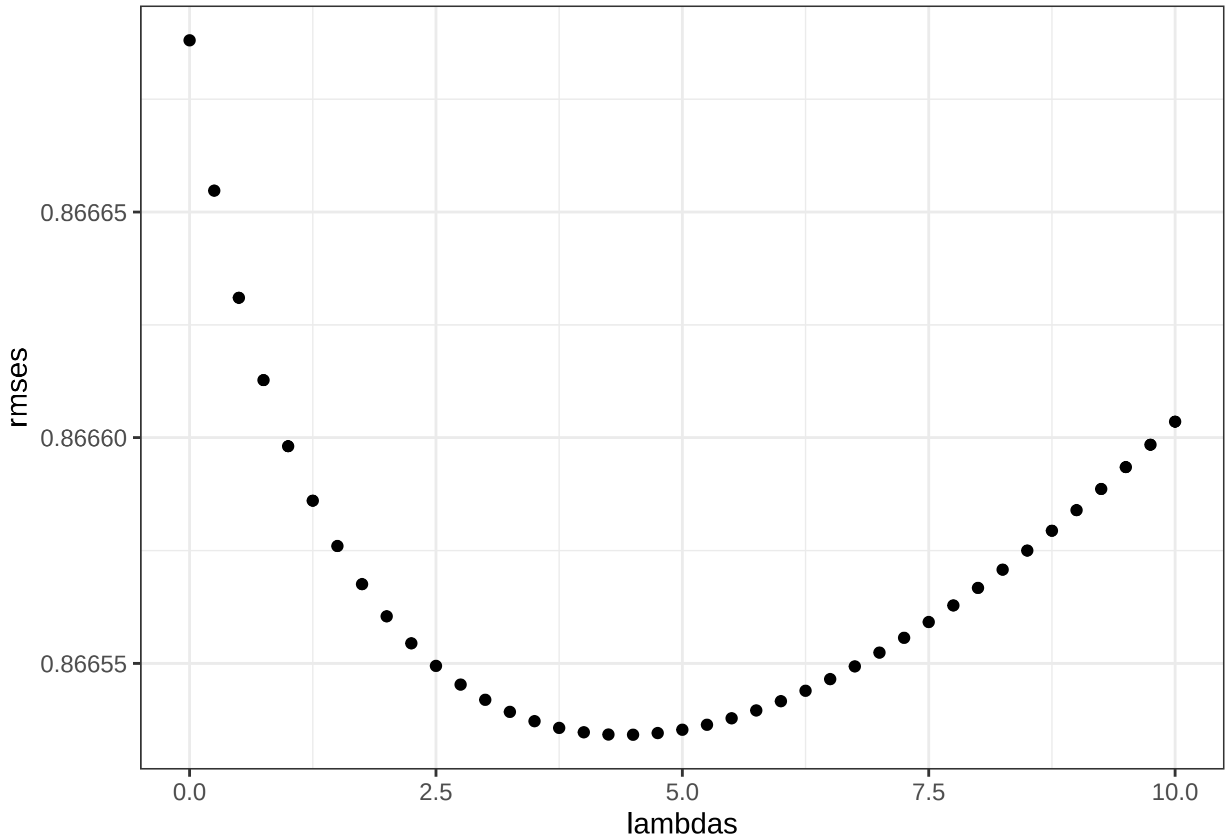
λ is a tuning parameter and I am going to use cross-validation to determine the value of λ that minimizes the RMSE.

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.

```r
# 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"
```

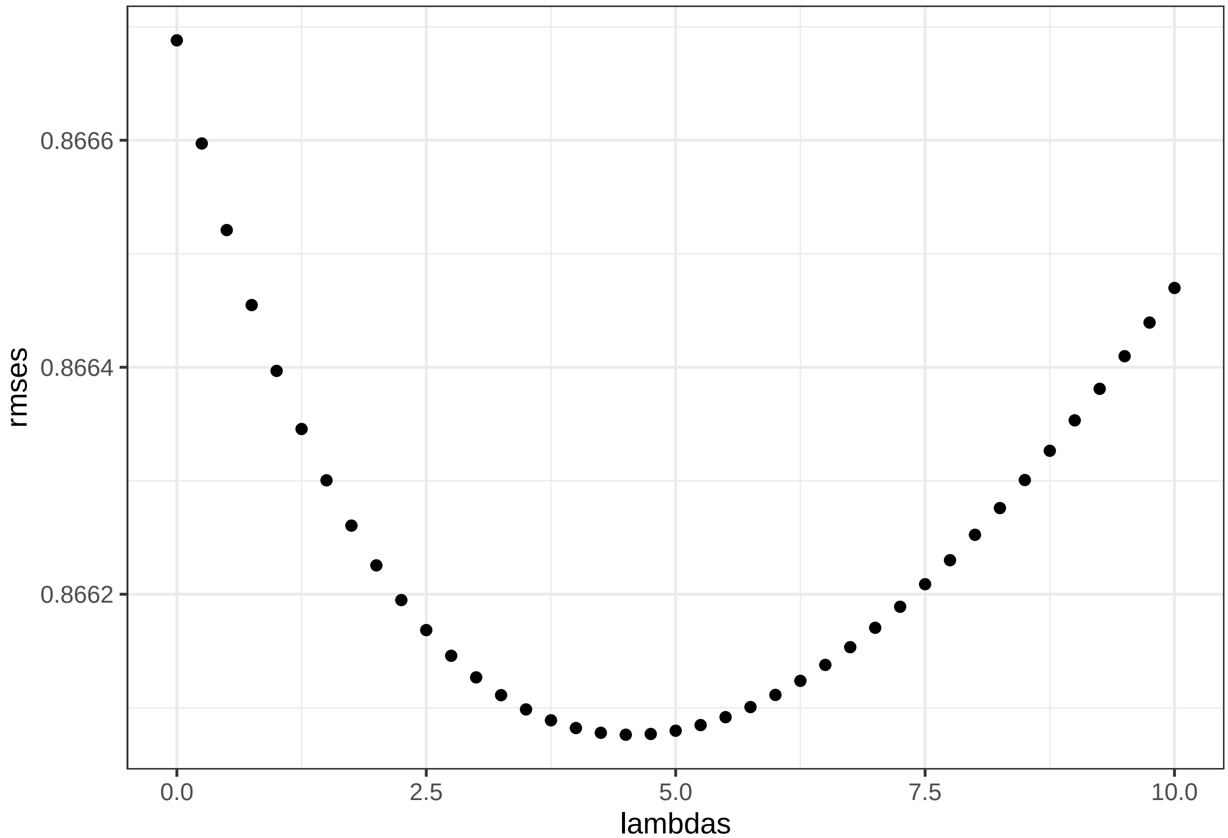
The RMSE improves a bit further compared to model 1. To do better I will try to regularize all the effects in the model. Let's apply a set of λ on the calculation of b_i, b_u, b_k and b_n and plot the obtained RMSEs.

```
# 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 makes sense because as we observed on the plots of residuals vs. number of ratings earlier, the movie effect was the most impacted by the low number of ratings for some movies.

Therefore, this analysis allowed me to design 3 models that I will now be able to apply to the **validation** set in order to assess their final performance.

III. Results

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

```r
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.

- First, we can note that with the most performing model I designed, the RMSE only improved by 0.12 % compared to the baseline model. This means that the movie and user effects already capture a large part of the variability in the ratings. The genre and year effects I added only brought a marginal improvement. This actually makes senses as genre and year are intrinsic features of the movie itself, thus quite redundant with the movie effect itself.
- Second, the lowest achieved RMSE of 0.86429 remains quite high in itself. It means that if I had to predict a rating for a given user-movie combination, on average I would be quite off - e.g. predicting a 4.3 for a combination that is actually worth 3.5.

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.

IV. 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.