MovieLens Project

Arturo Pérez

21/6/2020

1 Overview

The MovieLens dataset its a recopilation of over 20 million ratings for over 27,000 movies by more than 138,000 users compiled by GroupLens Research Lab, but in this project we work with a subset of about 10 million ratings for more than 10,000 movies and about 70,000 users. The 10% of the dataset was separated as a validation set and the rest was used as a training set. The dataset contains the following variables:

- **userId**: A number that identifies each user
- **movieId**: A number that identifies each movie
- **rating**: The rating given by the user to the movie from 0,5 to 5,0 in 0,5 increments
- **timestamp**: Seconds from Jan 01 1970 to the moment when the rating was made
- **title**: Commercial name of the film, followed by the release year enclosed in parentheses
- **genres**: List of genres to which the film belongs

Our objetive is to build a model that uses the information available in the dataset to make predictions for the ratings given by the users. We use the $RMSE$ between the predictions made and the true ratings as a measure of performance and we want to achive a value of 0.864900 or below.

We already worked with an even smaller subset of the MovieLens database in the Machiche Learning course and we built some prediction models, so we used the last (and better) model built in the course as a starting point for this project. The model uses the average rating, the regularized movie effect and the regularized user effect to make predictions. On that basis we built a second model including more predictors to obtain a better RMSE: the timestamp, the genres of the movie and the release year. We further improved the results with a third model having into account the differences in the range of the ratings given by each user. After building the model we optimized the parameters using crossvalidation over the entire training set. The final step was computing the predictions for the validation set and the corresponding RMSE.

2 Dataset preparation

A previous step to analysis is to download the database, obtain the relevant data and reserve part of the set for validation. This section is based on the code provided by EDX, with some tweaks to avoid downloading the database again if the files have been already downloaded to the working directory using the original EDX code. This will save some time and bandwidth. Please, note that this code was adapted for R 3.2.2. If you use a version of R higher than 3.5, you'll need to modify the *seed* instruction.

```
# Required libraries

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")
```

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

# Download files if they are not present in the working directory.

if(!file.exists("ml-10M100K/ratings.dat") | !file.exists("ml-10M100K/movies.dat")) {
  dl <- tempfile()
  download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
  unzip(dl, "ml-10M100K/ratings.dat")
  unzip(dl, "ml-10M100K/movies.dat")
  rm(dl)
}

# Read data from the downloaded files

ratings <- fread(text = gsub("::", "\t", readLines("ml-10M100K/ratings.dat")),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines("ml-10M100K/movies.dat"), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data

set.seed(1) # if using R higher than 3.5 , use `set.seed(1, sample.kind="Rounding")`
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(ratings, movies, test_index, temp, movielens, removed)
```

3 Analysis

We are going to split the *edx* set into a training and a test set in the same way we split the *movielens* set into *edx* and *validation* sets. This will allow us to test our models without using the validation set at all.

```r
# Split edx set into train_set and test_set

test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, 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)
```

We will also define the same function *RMSE* used in the Machine Learning course:

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

3.1 Model #1

We are going to use the *Regularized Movie + User Effect Model* built in the Machine Learning course as our first model. In this model we use the average rating plus biases associated with movies and users. The movie and user effects are regularized using a penalty λ.

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

We are going to code the model into a function that, given a training set, a test set and a penalty:

- computes the average rating μ and estimates the biases $\hat{b}_i$ and $\hat{b}_u$ using only the train set
- uses this results to generate the predictions for the test set.

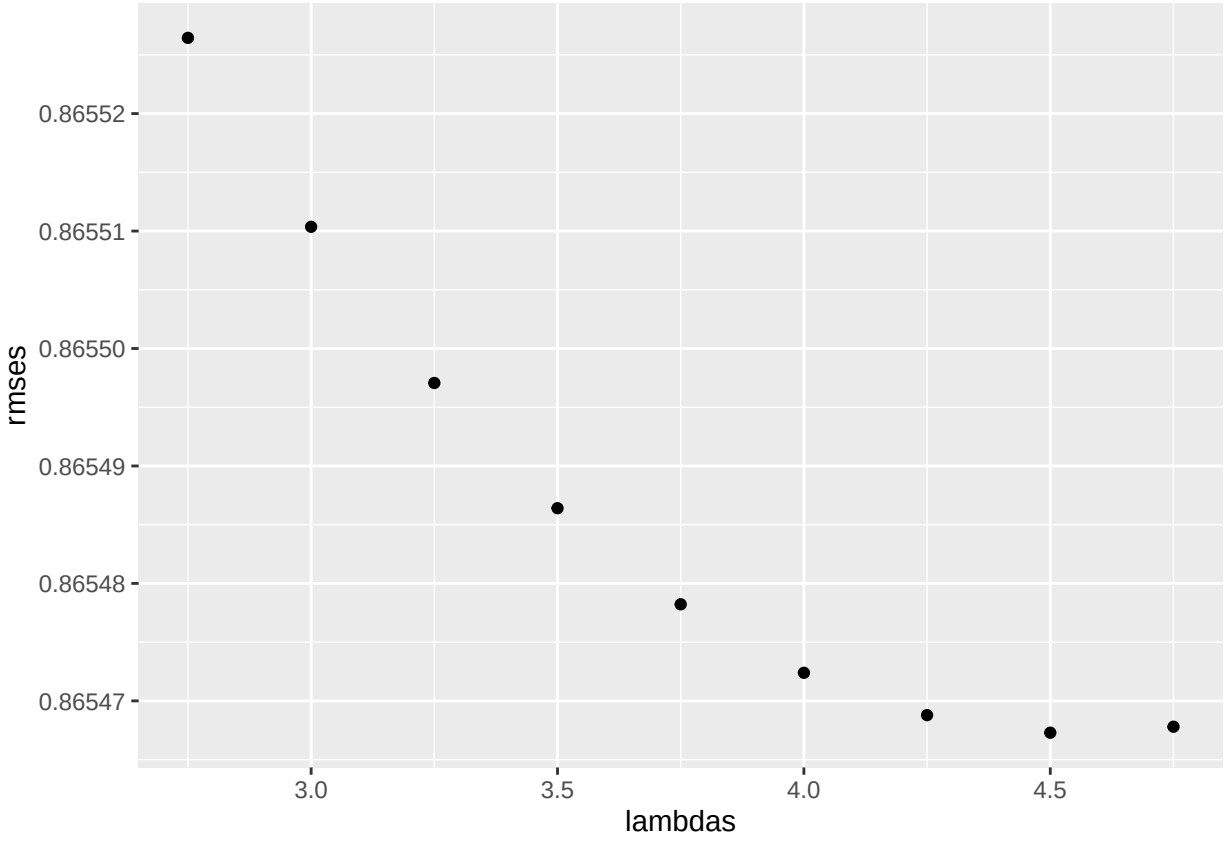
The biases are computed in the same way explained in the Machine Learning course: the regularized average after discounting the effect of previous predictors from the true rating.

```
# Model 1

model1 <- function(train_set, test_set, lambda) {

  # Overall average
  mu <- mean(train_set$rating)

  # Regularized movie effect
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

  # Regularized user effect
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

  # Prediction
  predictions <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
```

```
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return (predictions)
}
```

The optimal penalty found in the Machine Learning course for this model was $\lambda = 3.75$. We are going to run the same optimization using our training and test sets, but we will narrow the search to the vicinity of the optimal value found in the course.

```
# lambdas in the vicinity of 3.75
lambdas <- seq(2.75, 4.75, 0.25)

# Compute the RMSE for each lambda
rmses <- sapply(lambdas, function(lambda) {
  predictions <- model1(train_set, test_set, lambda)
  return(RMSE(predictions, test_set$rating))
})

# Plot the result
qplot(lambdas, rmses)
```



As we can see, our optimal penalty is $\lambda = 4.5$. We will store the RMSE computed for this model and this optimal penalty to compare it later with the other models.

```
# Display tibbles with 7 significant digits
options(pillar.sigfig = 7)
```

```
# Store the results of model #1
rmse_results <- tibble(
  method = "Model #1",
  RMSE = min(rmses),
  lambda = lambdas[which.min(rmses)]
)
rmse_results
```

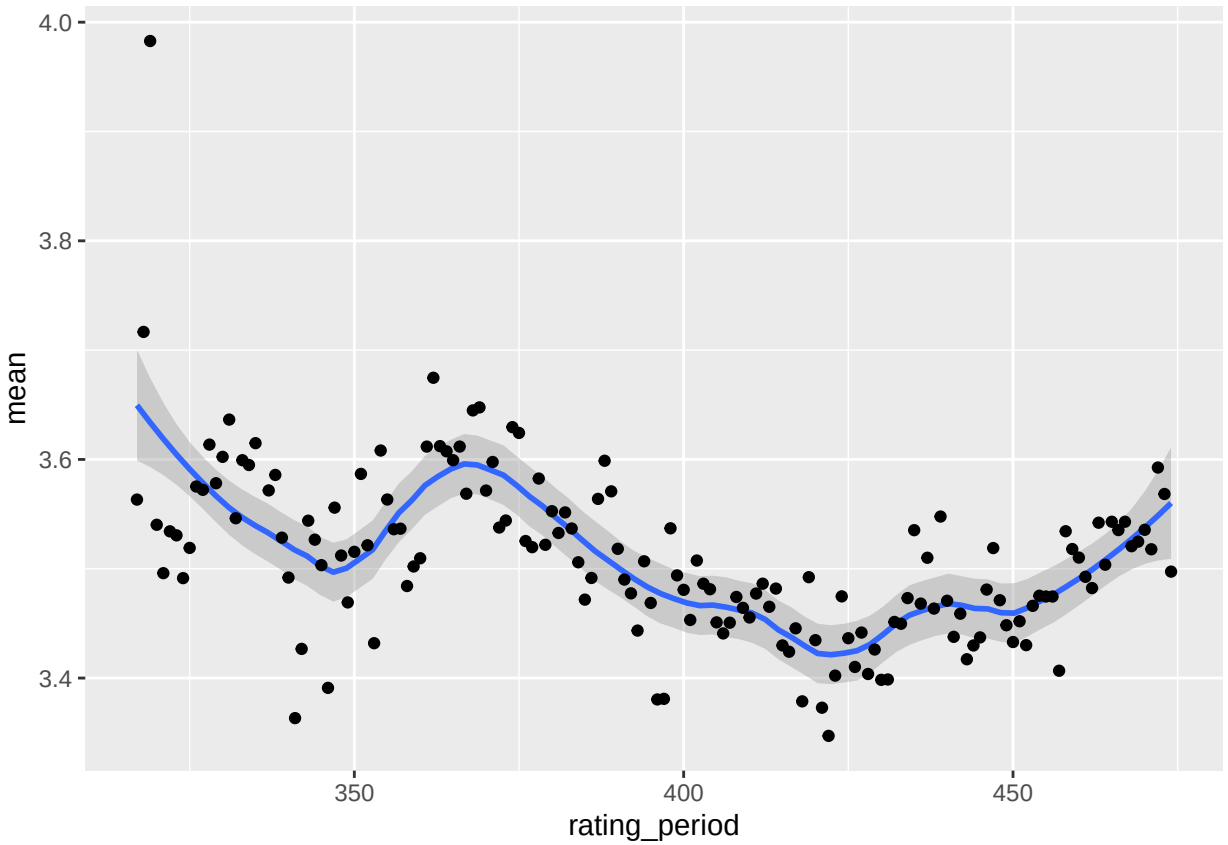
```
## # A tibble: 1 x 3
##   method        RMSE lambda
##   <chr>        <dbl>  <dbl>
## 1 Model #1 0.8654673    4.5
```

3.2 Model #2

We are going to improve our initial model adding more biases. To simplify code review and execution, all biases will be added in one step. The process of adding biases to the model is systematic and we would have to replicate a lot of code if we wanted to show the incorporation one by one.

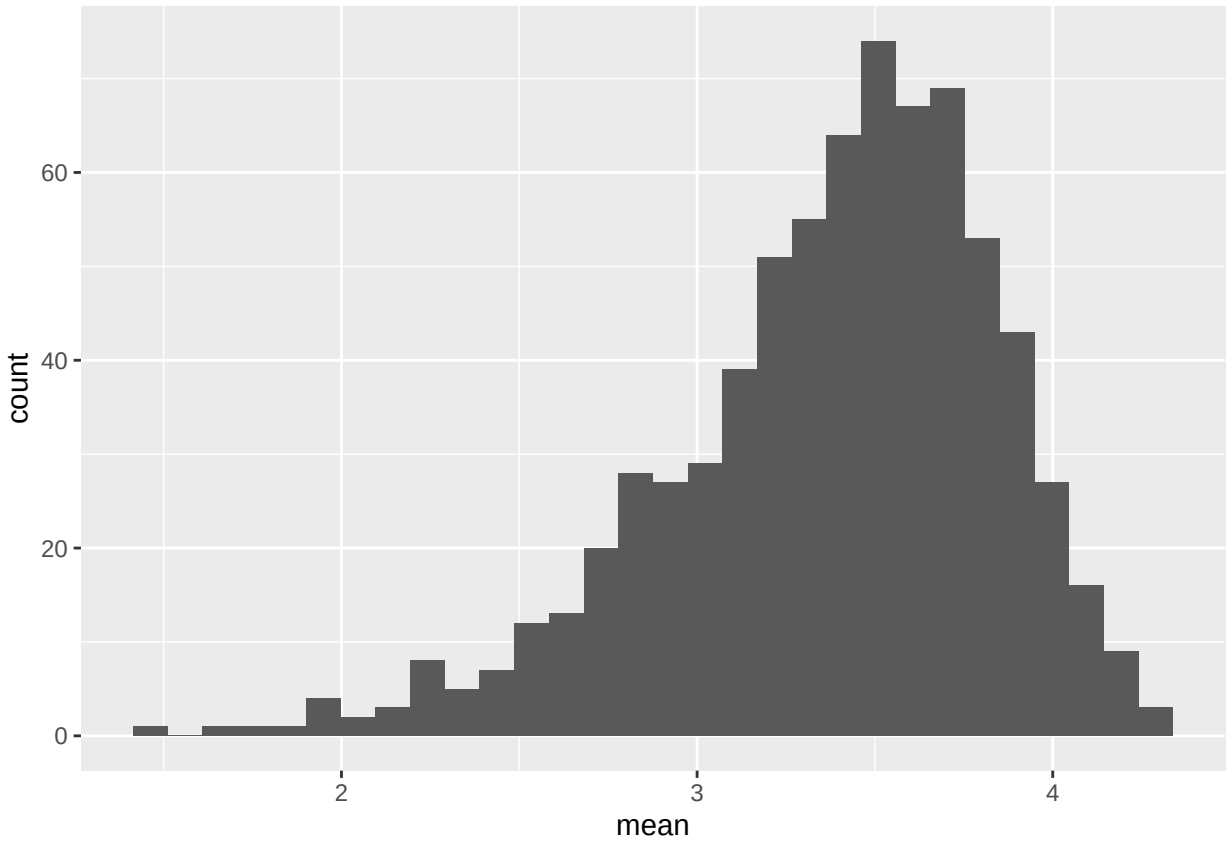
Let's start analysing the timestamp effect. In the following plot we show the variations of the ratings against the time when they were made. As we can see, there are fluctuations that can be captured having into account the timestamp. Each period in the plot covers 30 days, but we will use calendar months in our model.

```
# Plot average ratings over time
train_set %>%
  # 30 days * 24 hours/day * 3600 seconds/hour
  mutate(rating_period = floor(timestamp / (30*24*3600))) %>%
  group_by(rating_period) %>%
  summarize(mean = mean(rating), count = n()) %>%
  filter(count > 20) %>%
  ggplot(aes(rating_period, mean)) +
  geom_smooth(method='loess', span=0.3) +
  geom_point()
```



In the following figure we group the ratings by the genres of the films, and plot a histogram of the averages. There is a great dispersion, and that means there is a strong evidence of a genre effect that we can exploit adding the genres as a predictor.

```
# Histogram of averages when we group by genres
train_set %>%
  group_by(genres) %>%
  summarize(mean = mean(rating), count = n()) %>%
  filter(count > 20) %>%
  ggplot(aes(mean)) +
  geom_histogram(bins = 30)
```



The release year of the movie in encoded in the title so we are going to extract it using regular expressions. We are going to build a dataframe *release_years* contaning the movieId and the release year so we can join it to the datasets by movieId to obtain the release year instead of computing it each time:

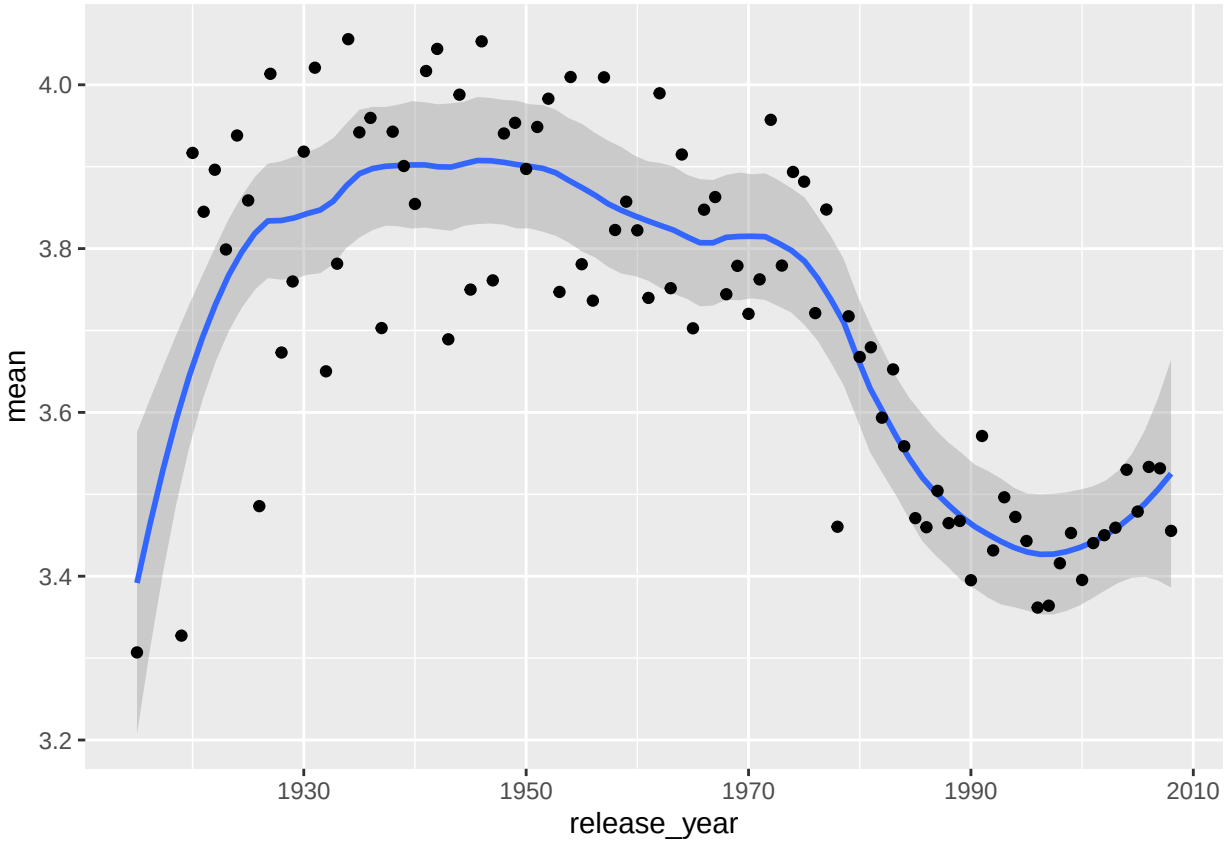
```
# Data frame with each movieId and release year
release_years <- train_set %>%
  distinct(movieId, title) %>%
  extract(title, c("title_tmp", "release_year"),
          regex = "^(.*)\\(([0-9]{4})\\)$", remove = F) %>%
  mutate(release_year = strtoi(release_year)) %>%
  select(movieId, release_year)

head(release_years)
```

```
##   movieId release_year
## 1     122         1992
## 2     185         1995
## 3     292         1995
## 4     329         1994
## 5     356         1994
## 6     362         1994
```

The following figure plots the average rating against the release year. We can see a strong pattern in which old movies have higher ratings than modern ones.

```
train_set %>%
  left_join(release_years, by='movieId') %>%
  group_by(release_year) %>%
  summarize(mean = mean(rating), count = n()) %>%
```

```
  filter(count > 100) %>%
  ggplot(aes(release_year, mean)) +
  geom_smooth(method='loess', span=0.3) +
  geom_point()
```



The three variables analyzed seem to provide relevant information, so we are going to include them in this second model:

$$Y_{u,i,t,g,y} = \mu + b_i + b_u + b_t + b_g + b_y + \varepsilon_{u,i,t,g,y}$$

where b_i is the movie effect, b_u the user effect, b_t the timestamp effect, b_g the genre effect and b_y the release year effect.

We will use the same approximation used in the Machine Learning course so we will estimate each factor $\hat{b}_x$ as the average of Y minus the combined effect of the previous factors. We will also regularize all terms using the same penalty:

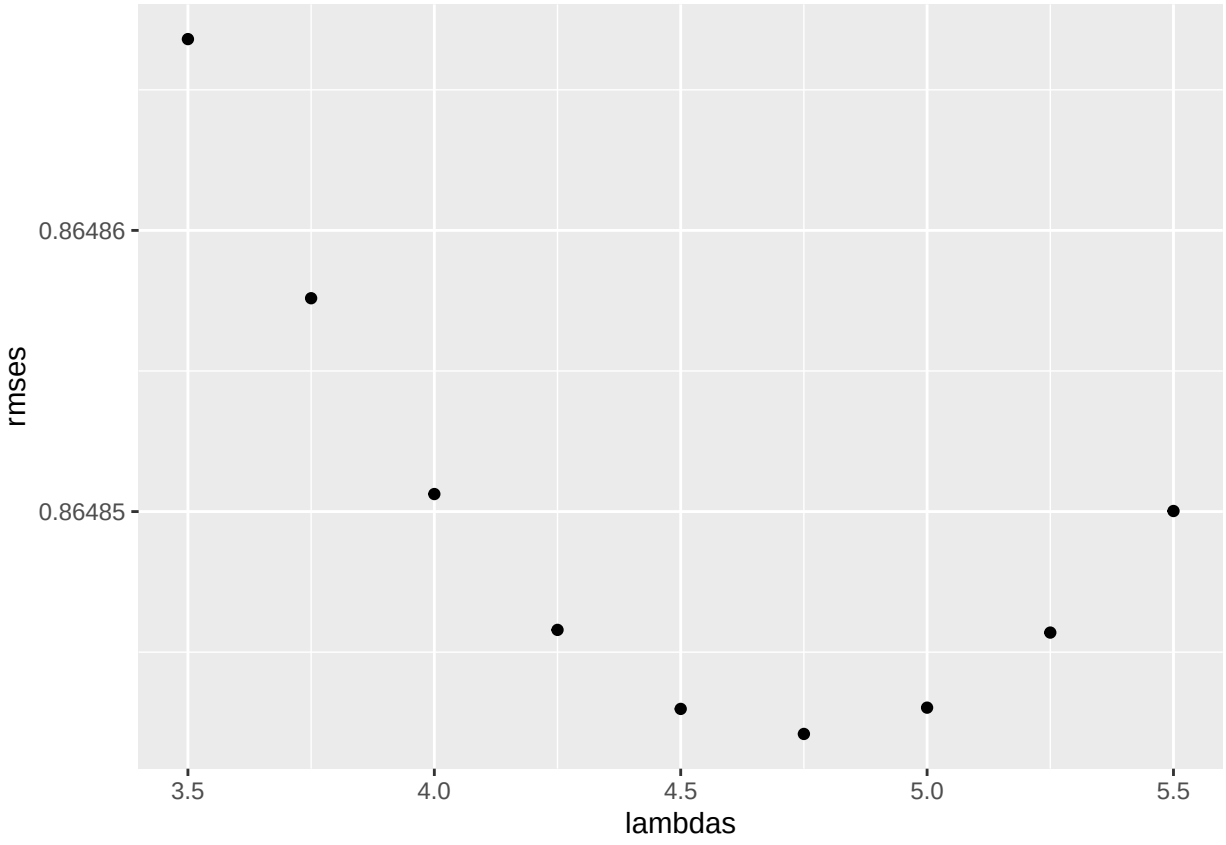
```
# We need the lubridate package to interpret the timestamp

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

# Model #2:

model2 <- function(train_set, test_set, lambda) {

  # Overall average
  mu <- mean(train_set$rating)
```

```r
# Regularized movie effect
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

# Regularized user effect
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

# Regularized rating month effect
b_t <- train_set %>%
  # The rating_period is YEAR-MONTH, ex: '2020-06'
  mutate(rating_period = format(as_datetime(timestamp), '%Y-%m')) %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(rating_period) %>%
  summarize(b_t = sum(rating - b_i - b_u - mu)/(n()+lambda))

# Regularized genre effect
b_g <- train_set %>%
  mutate(rating_period = format(as_datetime(timestamp), '%Y-%m')) %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_t, by="rating_period") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - b_t - mu)/(n()+lambda))

# Build the data frame (movieId, release_year) before computing the release_year effect.
# We are building it into this function because we are going to perform crossvalidation
# and the movies in the trainset will not be the same
release_years <- train_set %>%
  distinct(movieId, title) %>%
  extract(title, c("title_tmp", "release_year"),
          regex = "^(.*)\\(([0-9]{4})\\)$", remove = F) %>%
  mutate(release_year = strtoi(release_year)) %>%
  select(movieId, release_year)

# Regularized release year effect
b_y <- train_set %>%
  mutate(rating_period = format(as_datetime(timestamp), '%Y-%m')) %>%
  left_join(release_years, by="movieId") %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_t, by="rating_period") %>%
  left_join(b_g, by="genres") %>%
  group_by(release_year) %>%
  summarize(b_y = sum(rating - b_i - b_u - b_t - b_g - mu)/(n()+lambda))


# Prediction
predictions <-
```

```
    test_set %>%
    mutate(rating_period = format(as_datetime(timestamp), '%Y-%m')) %>%
    left_join(release_years, by = "movieId") %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "rating_period") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_y, by = "release_year") %>%
    mutate(pred = mu + b_i + b_u + b_t + b_g + b_y) %>%
    pull(pred)

  return (predictions)
}
```

Let's find the optimal penalty for this model:

```
# lambdas near 4.5
lambdas <- seq(3.5, 5.5, 0.25)

# compute the rmses
rmses <- sapply(lambdas, function(lambda) {
  predictions <- model2(train_set, test_set, lambda)
  return(RMSE(predictions, test_set$rating))
})

# and plot the result
qplot(lambdas, rmses)
```



As we can see, our optimal penalty in this case is $\lambda = 4.75$, and the $RMSE = 0.8648421$. The RMSE is slightly below our objective, but very close, so we are going to make one more improvement to our model to be more confident that we will reach the goal when tested over the validation set. Let's add the RMSE computed for this model to *rmse_results*.

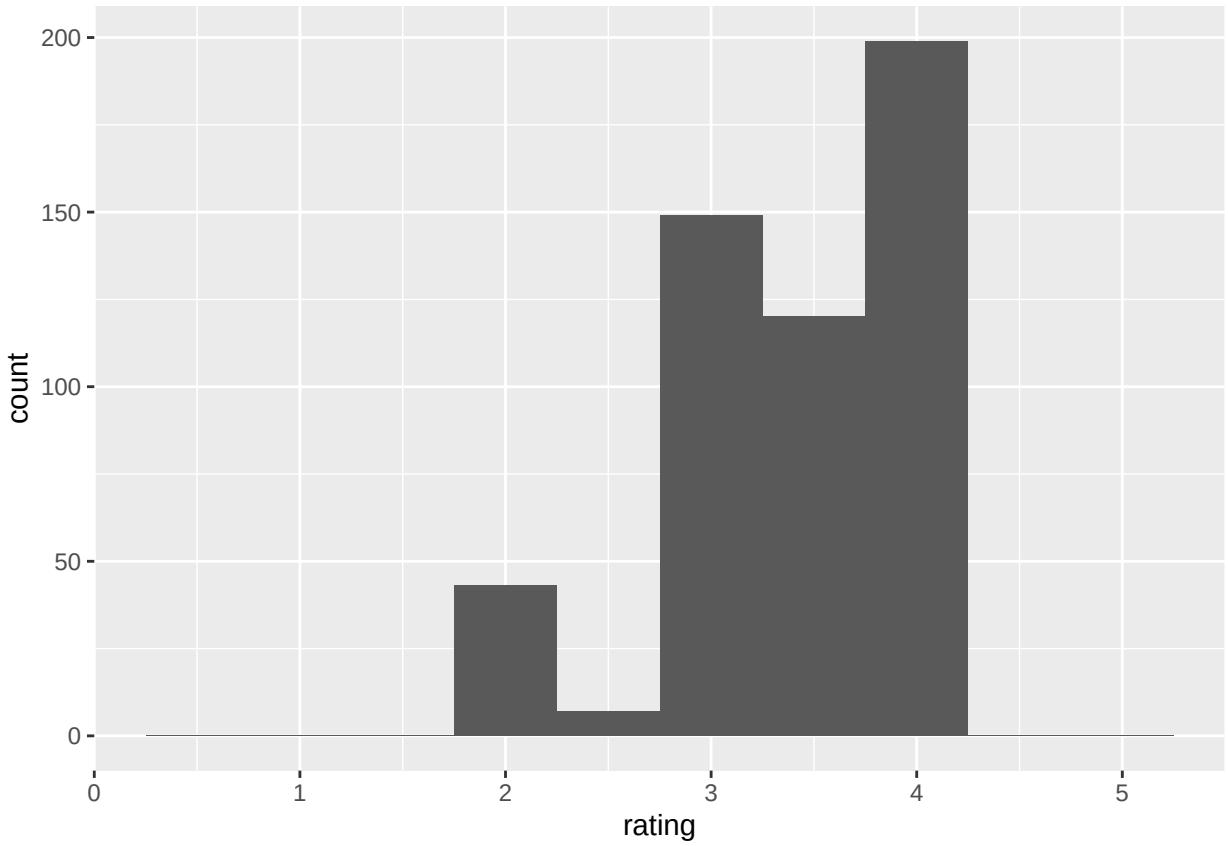
```
rmse_results <- bind_rows(rmse_results, tibble(
  method = "Model #2",
  RMSE = min(rmses),
  lambda = lambdas[which.min(rmses)]
))
rmse_results
```

```
## # A tibble: 2 x 3
##   method        RMSE lambda
##   <chr>        <dbl>  <dbl>
## 1 Model #1 0.8654673   4.5
## 2 Model #2 0.8648421   4.75
```

3.3 Model #3

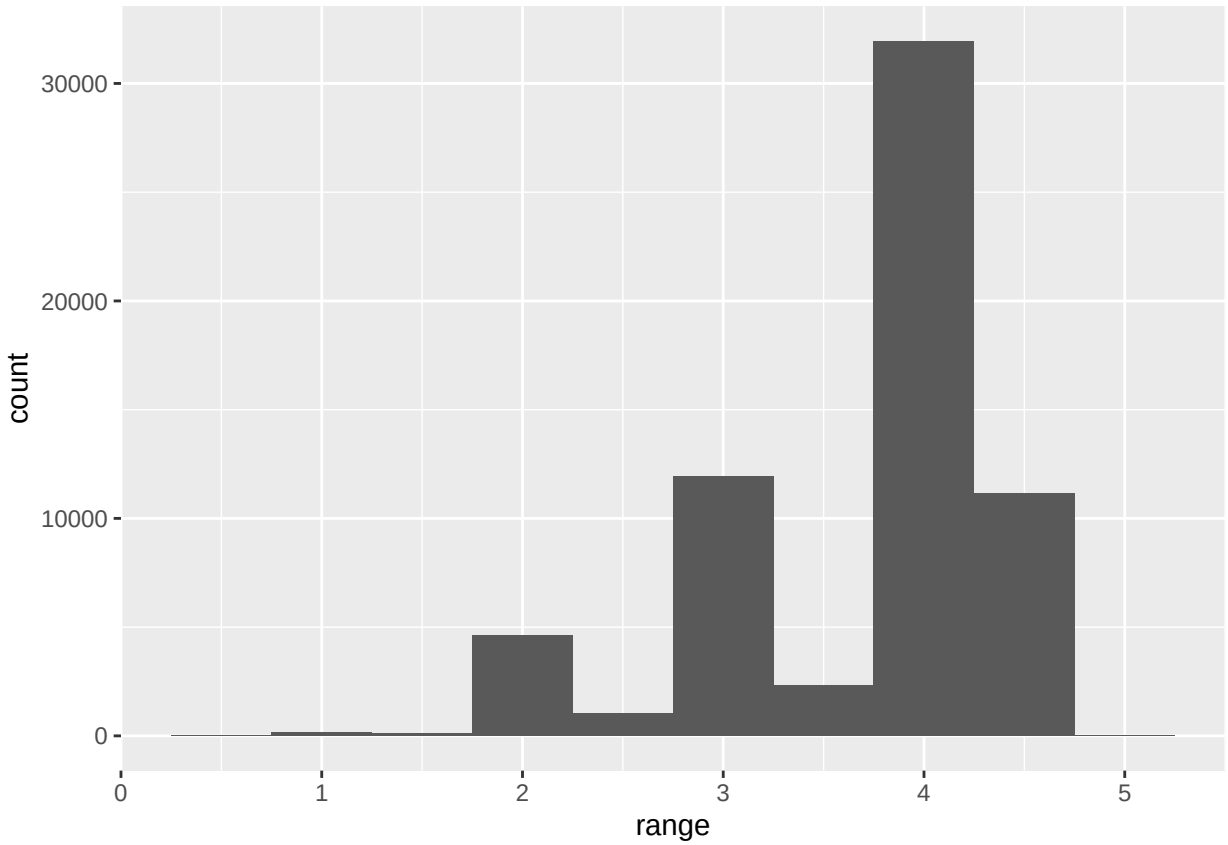
Until now we have only taken into account the mean value of the ratings when evaluating the effect of each factor. But there are also biases related to the deviation around the mean. As an example let's plot a histogram of the ratings of user 12760:

```
# Ratings given by user 12760
train_set %>%
  filter(userId == 12760) %>%
  ggplot(aes(rating)) +
  geom_histogram(bins=11, breaks = seq(0.25,5.25,0.5))
```



This user has rated 518 films and never gave a rating below 2 or above 4. There are people who never award the highest score probably because they consider that no movie is perfect. There are people who never give the minimum score, perhaps because they consider that every movie involves some effort, and a low rating (but not the worst) is sufficient punishment. Based on our training data, we can see that it is unusual for a user to use the full range of ratings available.

```
# Histogram of ranges
train_set %>%
  group_by(userId) %>%
  summarize(range = max(rating) - min(rating), count=n()) %>%
  filter(count>=20) %>%
  ggplot(aes(range)) +
  geom_histogram(bins=11, breaks = seq(0.25,5.25,0.5))
```



We are going to modify our model so the predicted ratings are delimited by the range of ratings given by the user in the training set. To do this we build a data frame with the minimum and maximum rating given by each user in the training set. The prediction is computed in the same way as in model #2 as a first step, and then we adjust the prediction, forcing its maximum and minimum value:

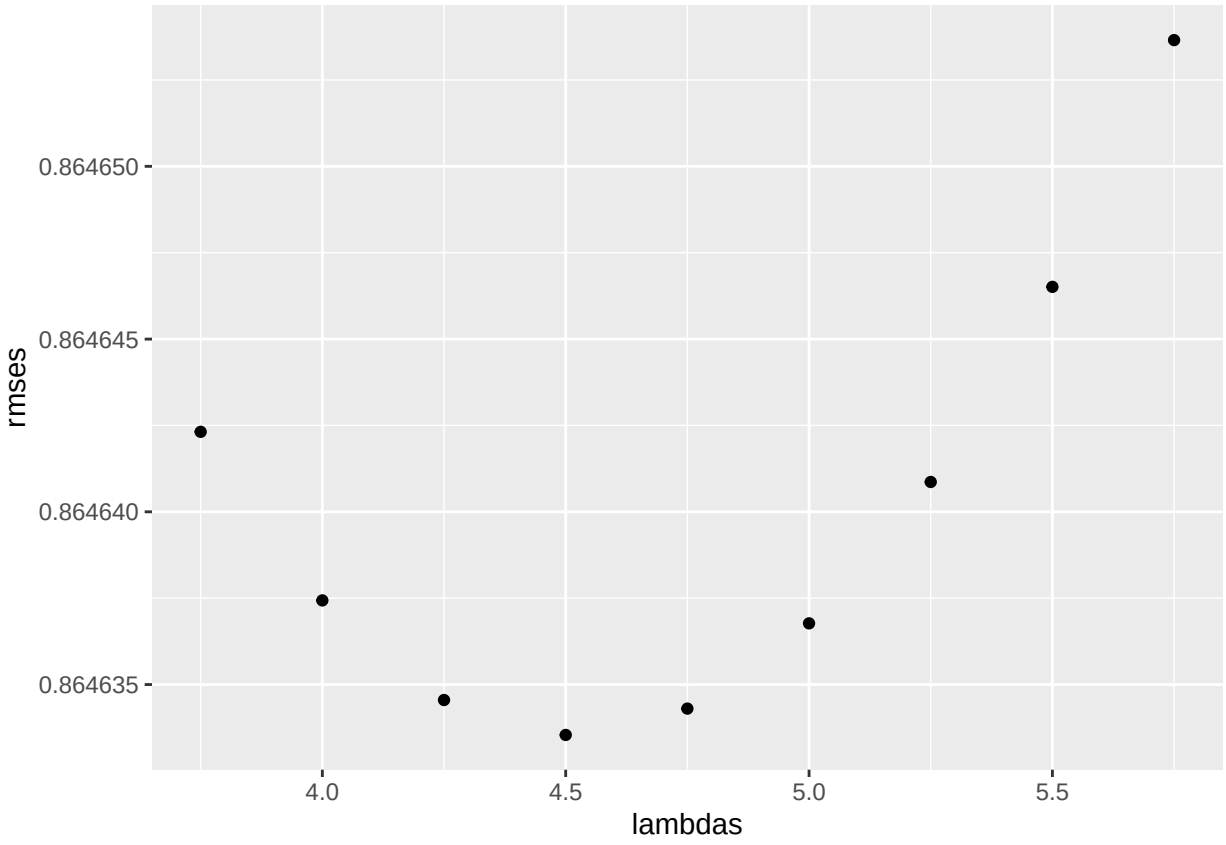
```
# Model 3

model3 <- function(train_set, test_set, lambda) {

  # Overall average
  mu <- mean(train_set$rating)

  # Regularized movie effect
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

  # Regularized user effect
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

  # Regularized rating month effect
  b_t <- train_set %>%
    mutate(rating_period = format(as_datetime(timestamp), '%Y-%m')) %>%
    left_join(b_i, by="movieId") %>%
```

```
  left_join(b_u, by="userId") %>%
  group_by(rating_period) %>%
  summarize(b_t = sum(rating - b_i - b_u - mu)/(n()+lambda))

# Regularized genre effect
b_g <- train_set %>%
  mutate(rating_period = format(as_datetime(timestamp), '%Y-%m')) %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_t, by="rating_period") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - b_t - mu)/(n()+lambda))

# Regularized release year effect
release_years <- train_set %>%
  distinct(movieId, title) %>%
  extract(title, c("title_tmp", "release_year"), regex = "^(.*)\\(([0-9]{4})\\)$", remove = F) %>%
  mutate(release_year = strtoi(release_year)) %>%
  select(movieId, release_year)

b_y <- train_set %>%
  mutate(rating_period = format(as_datetime(timestamp), '%Y-%m')) %>%
  left_join(release_years, by="movieId") %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_t, by="rating_period") %>%
  left_join(b_g, by="genres") %>%
  group_by(release_year) %>%
  summarize(b_y = sum(rating - b_i - b_u - b_t - b_g - mu)/(n()+lambda))

# THIS IS THE FIRST CHANGE INCLUDED IN THIS MODEL
# Minimum and maximum ratings per user
user_ranges <- train_set %>%
  group_by(userId) %>%
  summarise(min = min(rating), max=max(rating))

# Prediction
predictions <-
  test_set %>%
  mutate(rating_period = format(as_datetime(timestamp), '%Y-%m')) %>%
  left_join(release_years, by = "movieId") %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_t, by = "rating_period") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_y, by = "release_year") %>%
  # WE GET THE MAX AND MIN FOR EACH USER
  left_join(user_ranges, by = "userId") %>%
  mutate(
    # COMPUTE THE PREDICTION AS IN MODEL #2
    old_pred = mu + b_i + b_u + b_t + b_g + b_y,
    # AND THEN LIMIT THE RESULT BASED ON THE MIN AND MAX FOR THE USER
    pred = ifelse(old_pred > max, max, ifelse(old_pred < min, min, old_pred ))
```

```
    ) %>%
    pull(pred)

  return (predictions)
}
```

Let's find the optimal penalty for this model:

```
# lambdas near 4.75
lambdas <- seq(3.75, 5.75, 0.25)

# compute the rmses
rmses <- sapply(lambdas, function(lambda) {
  predictions <- model3(train_set, test_set, lambda)
  return(RMSE(predictions, test_set$rating))
})

# and plot the result
qplot(lambdas, rmses)
```



The optimal penalty for this model is $\lambda = 4.5$, and the $RMSE = 0.8646335$. Let's add the RMSE computed for this model to *rmse_results*.

```
rmse_results <- bind_rows(rmse_results, tibble(
  method = "Model #3",
  RMSE = min(rmses),
  lambda = lambdas[which.min(rmses)]
))
```

```
rmse_results
```

```
## # A tibble: 3 x 3
##   method        RMSE lambda
##   <chr>        <dbl>  <dbl>
## 1 Model #1 0.8654673   4.5
## 2 Model #2 0.8648421   4.75
## 3 Model #3 0.8646335   4.5
```

3.4 Parameter optimization

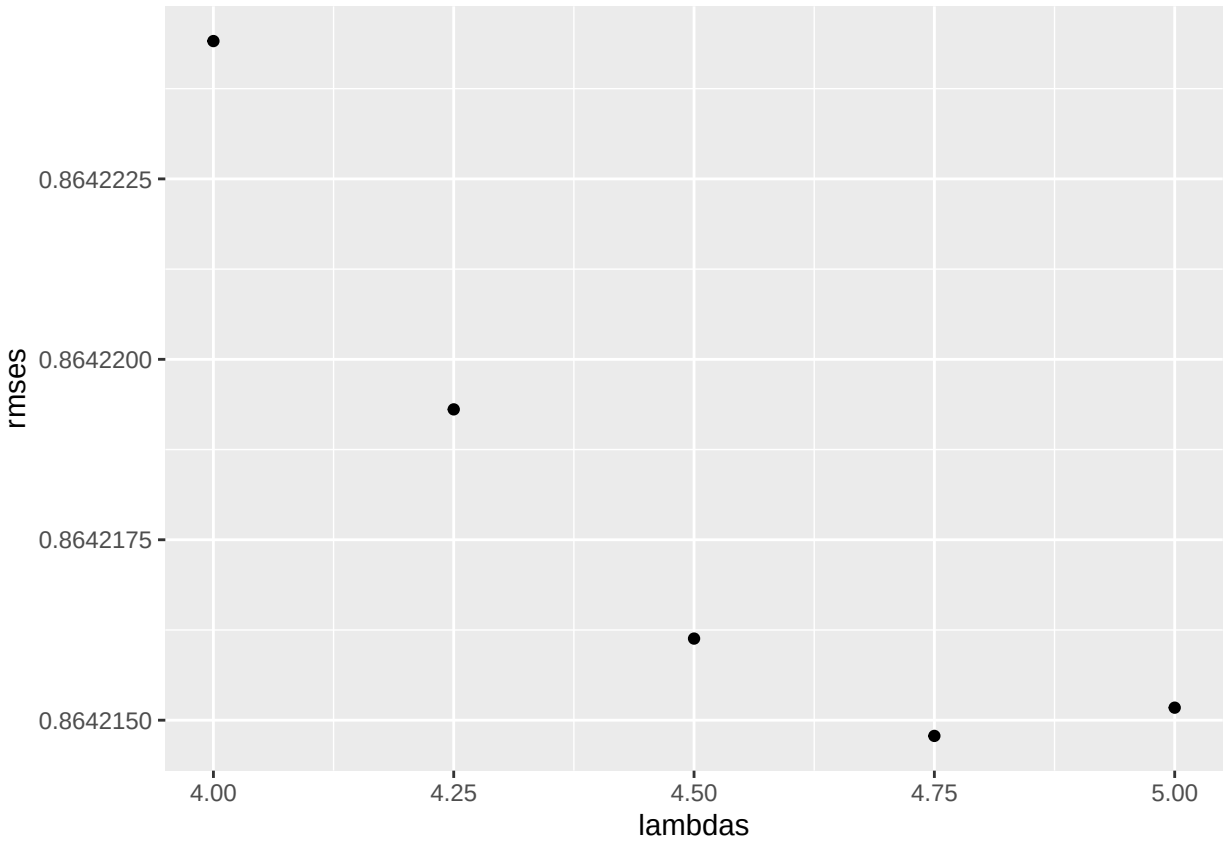
We are going to use model #3 as our final prediction model. We can use the entire *edx* dataset to fine tune the penalty using crossvalidation. We are going to split this dataset in 10 folds, so will reserve about 10% of the observations for testing in each fold, applying the same procedure used to create the *validation* set (remove movies and users that are not in the training set and merge the removed rows with the training set).

The dataset is sorted by userId (or at least they tend to be adjacent in the set) so we can't simply split the dataset into chuncks. If we do this we will see that there is a small intersection between the users included in the test set and the ones in the training set. To avoid this we build an index for all the rows and suffle it before splitting it in 10 chunks.

This block of code is very time consuming.

```
# lambdas near 4.5
lambdas <- seq(4, 5, 0.25)

# list of 10 test indexes
indexes <- split(sample(seq(1, nrow(edx))), seq(1,10))

# compute the rmse for each test index
rmses_folds <- sapply(indexes, function(index) {

  # Split edx into train and test sets using the current index
  test_index <- index
  train_set <- edx[-test_index,]
  temp <- edx[test_index,]
  test_set <- temp %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")
  removed <- anti_join(temp, test_set)
  train_set <- rbind(train_set, removed)
  rm(test_index, temp, removed)

  # Compute the RMSE for each penalty
  rmses_partition <- sapply(lambdas, function(lambda) {
    predictions <- model3(train_set, test_set, lambda)
    return(RMSE(predictions, test_set$rating))
  })

  # Return the RMSEs
  return(rmses_partition)

})
```

```r
# plot the mean rmse for each lambda
rmses <- rowSums(rmses_folds)/10
qplot(lambdas, rmses)
```



According to the result, we will finally use a penalty value of $\lambda = 4.75$. We will store the result in our *rmse_results* table.

```r
rmse_results <- bind_rows(rmse_results, tibble(
  method = "Model #3 after optimization",
  RMSE = min(rmses),
  lambda = lambdas[which.min(rmses)]
))
```

4 Results

The *rmse_results* table summarizes the evolution in the construction of a prediction model capable of achieving the proposed objective of a RMSE below 0.864900.

method	RMSE	lambda
Model #1	0.8654673	4.50
Model #2	0.8648421	4.75
Model #3	0.8646335	4.50
Model #3 after optimization	0.8642148	4.75

As we include more information in the model, the $RMSE$ decreases as expected. There is a jump in quality after the optimization using the entire *edx* set, but it is not due to a better selection of parameters. All the average rmses for all the lambdas evaluated in the crossvalidation are lower than the $RMSE$ obtained for model #3. This is simply because the initial partition of *edx* into a *train_set* and a *test_test* was, by chance, "harder" to predict for our model. The results obtained during crossvalidation give us a better idea of the performance we can expect from our model.

Let's now apply our final model to the *validation* set:

```
lambda <- 4.75
predictions <- model3(edx, validation, lambda)
RMSE(predictions, validation$rating)
```

```
## [1] 0.863978
```

Using the proposed model, we have achieved an RMSE of 0.863978, clearly below the initial target. As we can see, our model performs even better than expected. It may be due to chance (the information in this particular partition of the *movielens* database into the *edx* set and *validation* is better exploited by our model) or it may be due to the fact that we have more information available (we are using the entire *edx* set to make our predictions but during the analysis we were using only 90%).

5 Conclusion

In this project we have proposed some improvements to the prediction model developed during the Machine Learning course for the MovieLens dataset. Based on a model that uses the general average rating, the regularized movie effect and the regularized user effect, we have added three more predictors: the regularized average rating per rating period, the regularized average rating per genre and the regularized average rating per release year. We have seen that the user biases not only influences the mean of the ratings, but also influences the dispersion, so we have included this factor in our model as well.

These changes have allowed an improvement in the quality of the predictions enough to reach our goal of achieving a RMSE below 0.864900 in our validation set. But if we compare the performance of our initial model and the final model (before crossvalidation for a fair comparison) we can see an improvement of only about 0.1%. The variables added to the model seem promising but it looks like they were already taken into account to a large extent within the biases initially calculated in the movie and the user effect.

As future work, it would be interesting to explore the use of dimensional reduction techniques such as SVD. These techniques allow to alleviate to some extent the main problem in recommendation systems like the one we are analyzing, which is the sparsity of the ratings matrix, so they have a great potential to boost performance.