Movie Recommendation System Capstone Project

José Caloca

13/02/2020

Introduction

The aim of this project is to predict movie ratings across users, using the **MovieLens Dataset**. This dataset has 9,000,055 rows and 6 columns. The names of the variables (columns) are the following:

- **userId**: The class of userId is *integer*.
- **movieId**: The class of movieId is *numeric*.
- **rating**: The class of rating is *numeric*.
- **timestamp**: The class of timestamp is *integer*.
- **title**: The class of title is *character*.
- **genres**: The class of genres is *character*.

This dataset is composed by 10,677 different movies and 69,878 different users along the observations.

In this project we will be analysing different machine learning algorithms and comparing them so that we can select the one with the lowest Root-Mean-Square Error (RMSE).

Methods/Analysis

The first step is to load the libraries to be used, and to download the MovieLens dataset. For this we will run the following code that allows us to create the following datasets: **edx and validation**.

```
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")

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

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
```

```
# if using R 3.6 or earlier
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, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
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)
```

Once our datasets are loaded in our R environment, we can proceed to implement our machine learning algorithms and techniques. In this sense, we split the edx dataset into *test and training sets* using the **caret package**. Also, we use a semi-join to ensure that the movies and users in the training set are also in the test set.

We will be using the *train_set* for training our machine learning algorithms and the *test_set* will be only used for testing the performance of our algorithms.

```
test_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)
test_set <- edx[test_index,]
train_set <- edx[-test_index,]

test_set <- test_set %>% semi_join(train_set, by = "movieId")
test_set <- test_set %>% semi_join(train_set, by = "userId")
```

Missing values tend to affect the results of our estimations, therefore we are insterested in detecting any potential missing value in the dataset using the following code.

```
sum(is.na(train_set$rating))
```

```
## [1] 0
```

This reflects that the are no missing values in the *train_set* dataset and we can proceed with our analysis.

As the model will be assessed based on the RMSE, we create a function that calculates this value.

```
RMSE <- function(actual, predicted){
sqrt(mean((actual - predicted)^2))
}
```

Initial models

Movie effects

Our first model is based on the movies effects. As many movies are rated differently by the users, this model will be adjusted for that. We create the first model as follows:

```
library(tidyverse)
mu <- mean(train_set$rating)

movie_effects <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# We make the prediction using the mean of movie ratings and movie effects

pred_1 <- mu + test_set %>%
  left_join(movie_effects, by = "movieId") %>%
  pull(b_i)

rmse1 <- RMSE(test_set$rating, pred_1)
rmse1
```

```
## [1] 0.9437144
```

This first basic model yields a RMSE of 0.9437144.

Movie + User Effects

With the intention to improve the previous model and lower the RMSE obtained, we present a new model that takes into consideration the user's preferences. The code used is the following:

```
user_effects <- train_set %>%
  left_join(movie_effects, by = "movieId") %>%
  #we need to join the tables in order to have access to b_i
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

pred_2 <- mu + test_set %>%
  left_join(movie_effects, by = "movieId") %>%
  left_join(user_effects, by = "userId") %>%
  mutate(bu_bi = b_u + b_i) %>%
  pull(bu_bi)

rmse2 <- RMSE(test_set$rating, pred_2)
rmse2
```

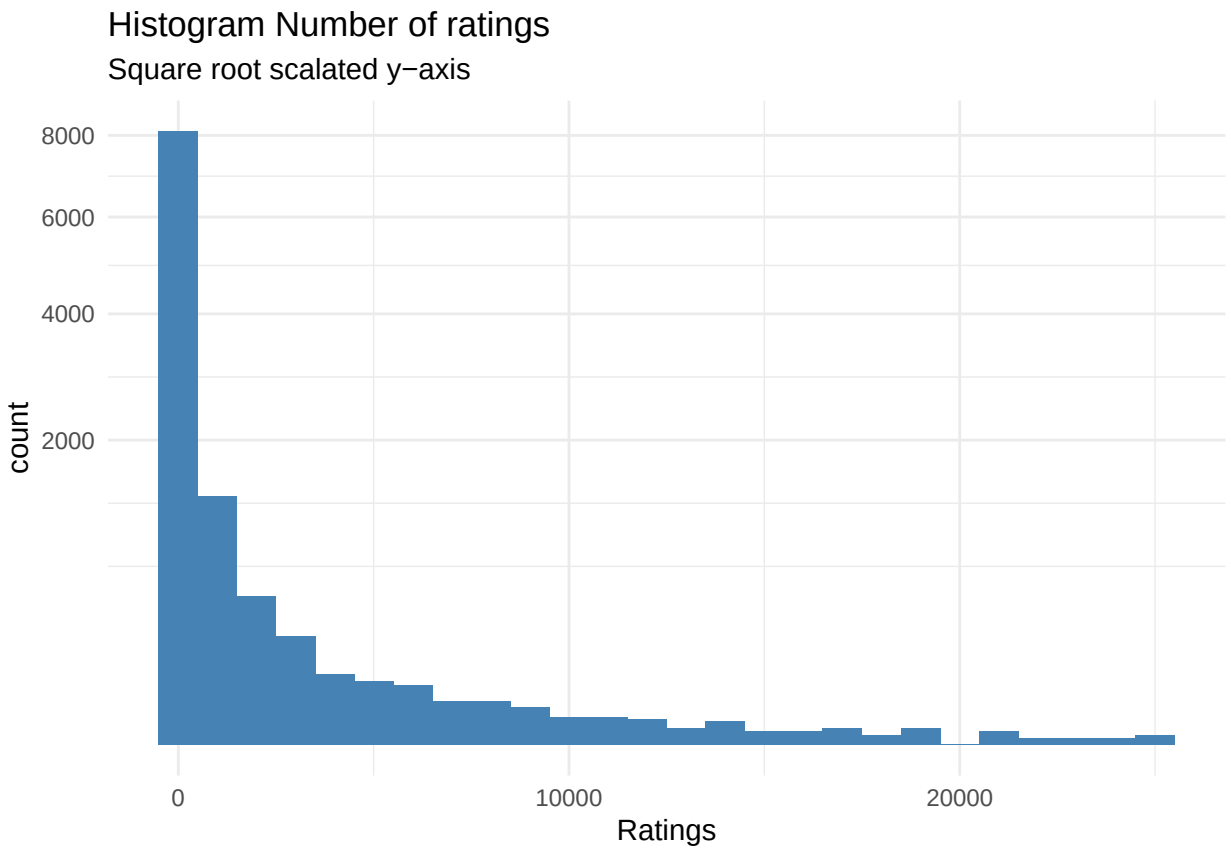
```
## [1] 0.8661625
```

This augmented model by user effects yields a RMSE of 0.8661625. We can see a considerable improvement with respect to the first model.

Regularised Models

Regularised Movie Effects

In order to understand this dataset, it is fundamental to see how movies are rated. The following graph shows a histogram of the number ratings for each unique film.



From the above histogram we can visualise that many movies are not rated equally, in fact, many movies have more ratings than others. To account for this, we will use regularization by movie effects.

We use the regularization method that minimises the error and improves our estimates by adding a penalty term which helps to fit our model and avoids overfitting. The value that minimizes this estimate is λ, and this value of λ that optimises our model is not yet known, so we use cross validation to find the λ that minimises error of our model. It is appropriate to use regularization because some movies were given great ratings very few times.

```
lambdas <- seq(0,10,0.1)

sums <- train_set %>%
        group_by(movieId) %>%
        summarize(s = sum(rating - mu), nums = n())
```

```
rmse_cv <- sapply(lambdas, function(lam){
#this function makes predictions using different values of lambda
      predictions <- test_set %>%
        left_join(sums, by = "movieId") %>%
        mutate(b_i = s/(nums + lam)) %>%
        mutate(preds = mu + b_i) %>%
        pull(preds)
      return(RMSE(test_set$rating, predictions))
})

lambda <- lambdas[which.min(rmse_cv)]
lambda
```

```
## [1] 1.7
```

By using the cross-validation technique we obtain the value of λ that optimises our model, which is $\lambda = 1.7$. Now tune our model with the value of λ obtained.

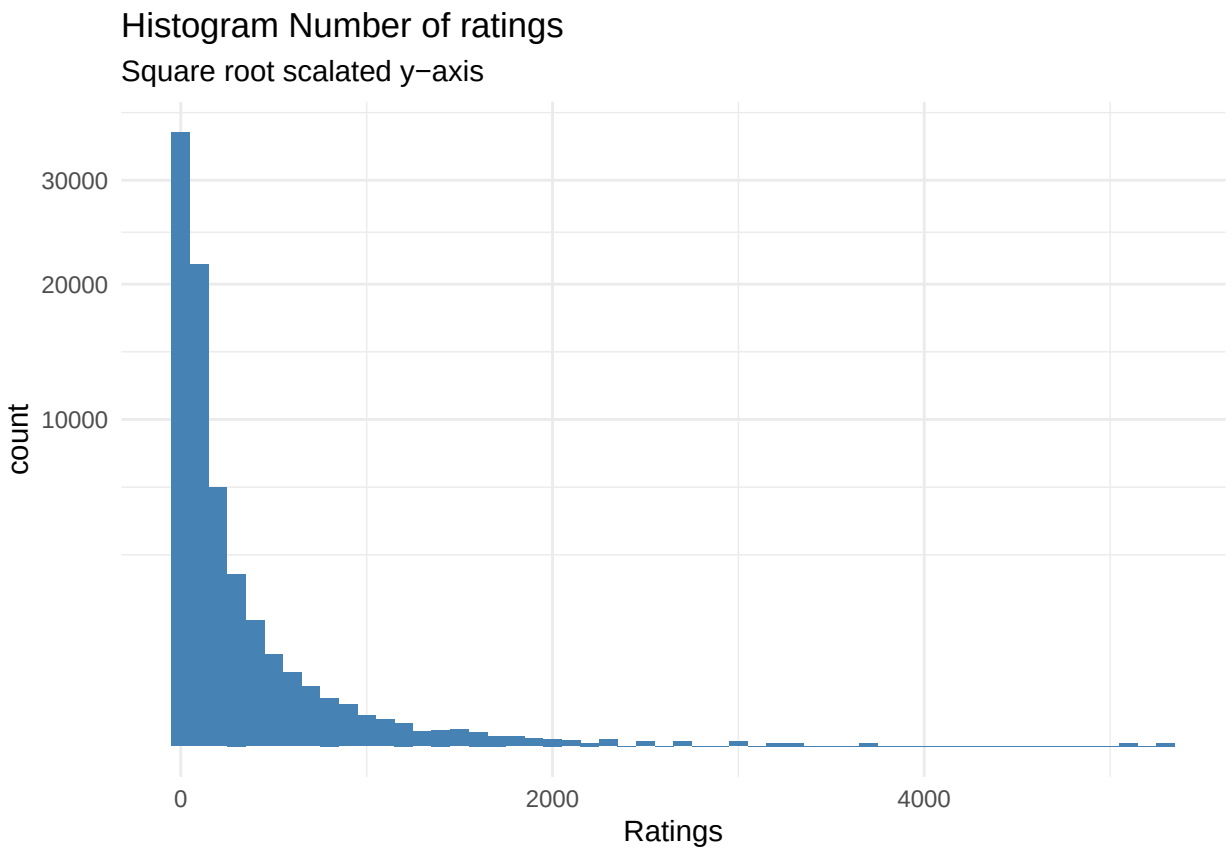
```
reg_movie_effects <- train_set %>%
  group_by(movieId) %>%
  summarize(reg_bi = sum(rating - mu)/(n() + lambda))

pred_3 <- test_set %>%
  left_join(reg_movie_effects, by = "movieId") %>%
  mutate(preds = mu + reg_bi) %>%
  pull(preds)

rmse3 <- RMSE(test_set$rating, pred_3)
rmse3
```

```
## [1] 0.9436775
```

By following this method, the above model yields a RMSE of 0.9436775. This model didn't improve more as expected. Now we will regularize by the user effect to see if there is any difference in the RMSE.

Regularized Movie + User Effects

For this we will present a histogram in which can be appreciated how users rate movies.



We will continue to use the regularization method as many users did not rate movies with the same frequency. It means that some users rated more movies than others. For this we have to look the optimal value of λ for our augmented model with movie and user effects as follows:

```
rmses_cv2 <- sapply(lambdas, function(lam){
  reg_movie <- train_set %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(lam + n())) #reg_movie will contain bi

  reg_user <- train_set %>%
    left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
    group_by(userId) %>%
    summarize(bu = sum(rating - mu - bi)/(lam + n()))

  predictions <- test_set %>%
    left_join(reg_movie, by = "movieId") %>%
    #need to join so we have access to both bi and bu
    left_join(reg_user, by = "userId") %>%
    mutate(preds = mu + bi + bu) %>%
    pull(preds)

  return(RMSE(test_set$rating, predictions))
})
lambda2 <- lambdas[which.min(rmses_cv2)]
lambda2
```

```
## [1] 4.7
```

By using the cross-validation technique we obtain the value of λ that optimizes our model, which is $\lambda = 4.7$. Now we will tune our model with the value of λ obtained.

```
#now we run the model with lambda = 4.7 to get the rmse.
reg_movie <- train_set %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(lambda2 + n())) #reg_movie will contain bi

reg_user <- train_set %>%
  left_join(reg_movie, by = "movieId") %>%
  #need to join so that we have access to bi
  group_by(userId) %>%
  summarize(bu = sum(rating - mu - bi)/(lambda2 + n()))

pred_4 <- test_set %>%
  left_join(reg_movie, by = "movieId") %>%
  #need to join so we have access to both bi and bu
  left_join(reg_user, by = "userId") %>%
  mutate(preds = mu + bi + bu) %>%
  pull(preds)

rmse4 <- RMSE(test_set$rating, pred_4)
rmse4
```

```
## [1] 0.8655424
```

This is model yields a RMSE of 0.8655424, which the lowest so far.

Ensembled model

We create an ensembled model using the predictions of the 4 previous analyzed models:

- movie effects.
- movie + user effects.
- regularized movie effects.
- regularized movie + user effects.

The approach to this in the following:

```
pred_5 <- (pred_1 + pred_2 + pred_3 + pred_4)/4

rmse5 <- RMSE(test_set$rating, pred_5)
rmse5
```

```
## [1] 0.8847612
```

This model yields a RMSE of 0.8847612. The ensemble model didn't improve much as expected.

Regularized Movie + User + Genre Effects

So far the regularized movie + user effects provided the best results in terms of RMSE. We assume that by including the regularized genre effect, it will improve the RMSE as some genres are watched and favored more than others, therefore there must be some influence based on genre. As before, we start by using cross-validation to find the optimal λ for this model.

```
rmses_cv3 <- sapply(lambdas, function(lam){
  reg_movie <- train_set %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(lam + n())) #reg_movie will contain bi

  reg_user <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
    #need to join so that we have access to bi
    group_by(userId) %>%
    summarize(bu = sum(rating - mu - bi)/(lam + n()))

  reg_genre <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - bi - bu)/(lam + n()))

  predictions <- test_set %>%
    left_join(reg_movie, by = "movieId") %>%
    #need to join so we have access to both bi and bu
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    mutate(preds = mu + bi + bu + b_g) %>%
    pull(preds)

  return(RMSE(test_set$rating, predictions))
})
lambda3 <- lambdas[which.min(rmses_cv3)]
lambda3
```

```
## [1] 4.7
```

The λ that minimizes the RMSE of the Regularized Movie + User + Genre Effects model is 4.7. Now we will tune our model using the value of λ obtained.

```
# now we run the model with the optimal value of lambda
reg_movie <- train_set %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(lambda3 + n())) #reg_movie will contain bi

reg_user <- train_set %>%
  left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
  group_by(userId) %>%
  summarize(bu = sum(rating - mu - bi)/(lambda3 + n()))

reg_genre <- train_set %>%
```

```
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - bi - bu)/(lambda3 + n()))

pred_6 <- test_set %>%
  left_join(reg_movie, by = "movieId") %>% #need to join so we have access to both bi and bu
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  mutate(preds = mu + bi + bu + b_g) %>%
  pull(preds)

rmse6 <- RMSE(test_set$rating, pred_6)
rmse6
```

```
## [1] 0.8655406
```

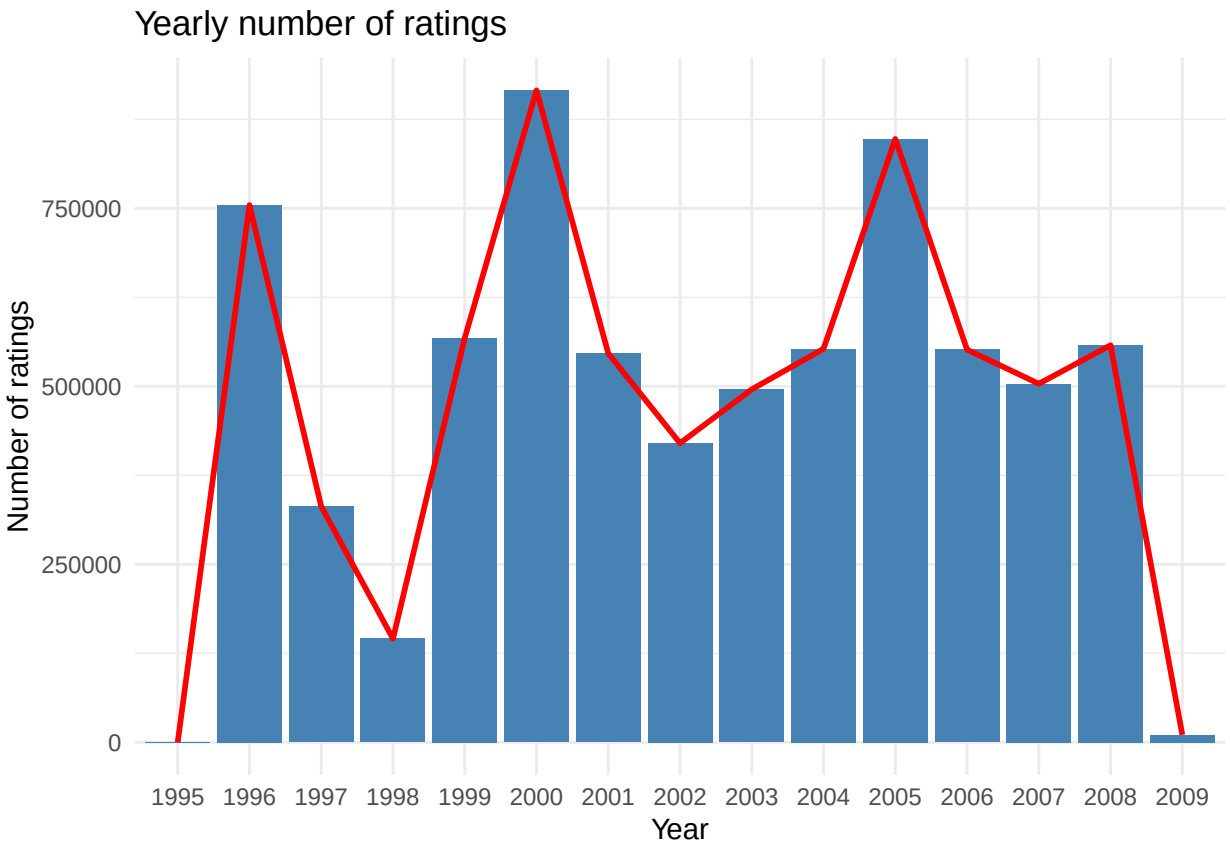
This model has a RMSE of 0.8655406 which is lower than the RMSE of the Regularized Movie + User Effects model. Although is not what it is expected to achieve this model, it yields very good results. Therefore we check the results after regularizing by year.

Regularized Movie + User + Genre + Year Effects

Although the previous model is better than the regularized movie + user effects one, We try to check if regularizing by month and year would make a difference. For this, we create a month and year column for the test and training set using the **Lubridate package**.

```
library(lubridate)
train_set <- train_set %>%
  mutate(date = as_datetime(timestamp),
         year = year(date),
         month = month(date))

test_set <- test_set %>%
  mutate(date = as_datetime(timestamp),
         year = year(date),
         month = month(date))
```

The following bar chart depicts the number of ratings per year and their variability during the timespan.



The variability in the number of ratings per year is defined by the standard deviation of the yearly number of ratings:

```
train_set %>% group_by(year = factor(year)) %>%
    summarise(number_ratings = n()) %>% pull(number_ratings) %>%
    sd()
```

```
## [1] 269639.7
```

This implies that the data changed in average 269639.7 ratings each year from 1995 to 2009. To this, we use cross-validation to find the optimal λ for the regularized movie + user + genre + year effects.

```
lambdas <- seq(0,10,0.25)
rmses_cv4 <- sapply(lambdas, function(lam){
  reg_movie <- train_set %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(lam + n())) #reg_movie will contain bi

  reg_user <- train_set %>%
    left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
    group_by(userId) %>%
    summarize(bu = sum(rating - mu - bi)/(lam + n()))

  reg_genre <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
```

```
    left_join(reg_user, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - bi - bu)/(lam + n()))

  reg_year <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - bi - bu - b_g)/(lam + n()))

  predictions <- test_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    left_join(reg_year, by = "year") %>%
    mutate(preds = mu + bi + bu + b_g + b_y) %>%
    pull(preds)

  return(RMSE(test_set$rating, predictions))
})

lambda <- lambdas[which.min(rmses_cv4)]
lambda
```

```
## [1] 4.75
```

the λ that minimizes the RMSE of the regularized movie + user + genre + year effects model is 4.75. Now we tune of model with the value of λ obtained.

```
reg_movie <- train_set %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(lambda + n())) #reg_movie will contain bi

reg_user <- train_set %>%
  left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
  group_by(userId) %>%
  summarize(bu = sum(rating - mu - bi)/(lambda + n()))

reg_genre <- train_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - bi - bu)/(lambda + n()))

reg_year <- train_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - bi - bu - b_g)/(lambda + n()))

pred_7 <- test_set %>%
```

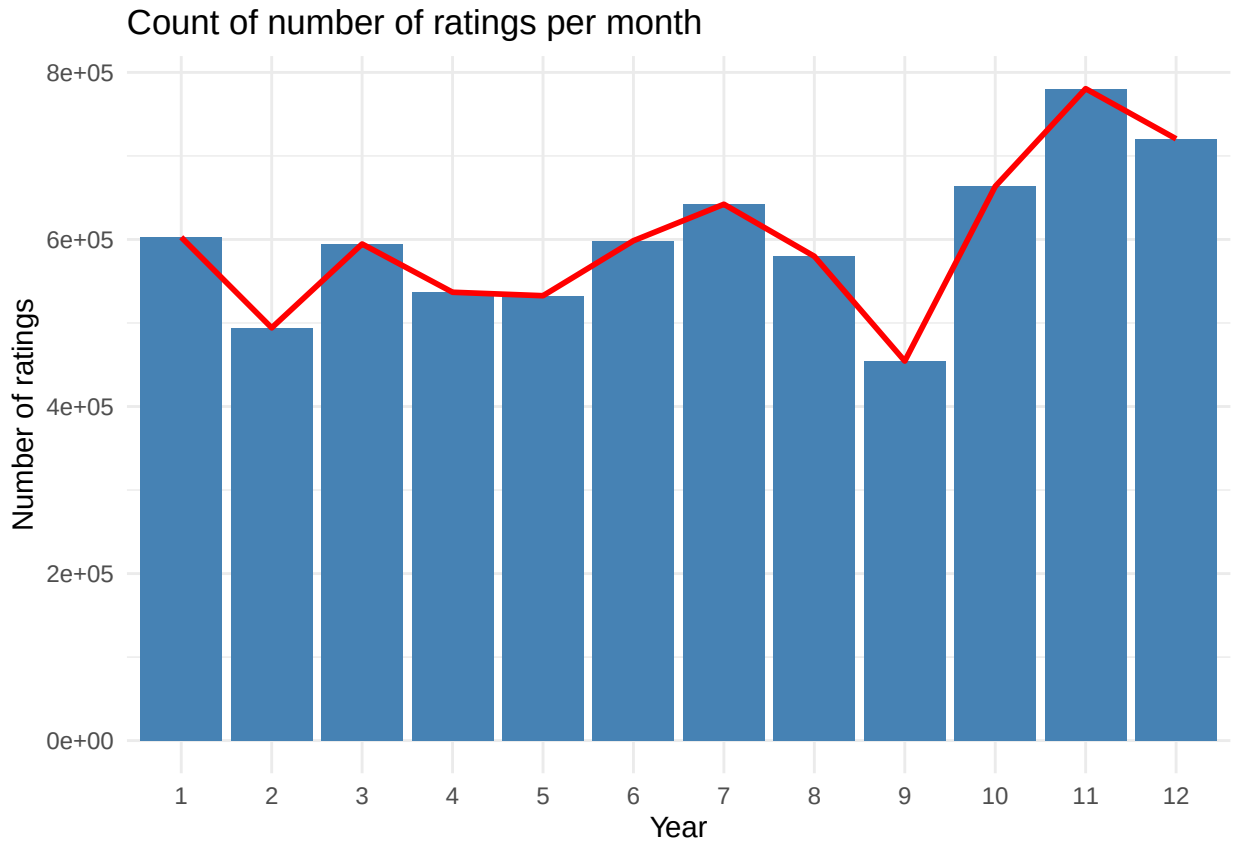
```
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  left_join(reg_year, by = "year") %>%
  mutate(preds = mu + bi + bu + b_g + b_y) %>%
  pull(preds)

rmse7 <- RMSE(test_set$rating, pred_7)
rmse7
```

```
## [1] 0.8655207
```

This model yields a RMSE of 0.8655207, which is much better than the previous model. Therefore we check the model by adding the variable *month*.

Regularized Movie + User + Genre + Year + Month Effects

We analyze the variability of the number of ratings per month. The following bar chart shows the sum of ratings per month and their variability during the timespan



The variability in the number of ratings per year is defined by the standard deviation of the monthly number of ratings:

```
train_set %>% group_by(year = factor(month)) %>%
    summarise(number_ratings = n()) %>% pull(number_ratings) %>%
    sd()
```

```
## [1] 92581.21
```

This implies that the data changed in average 92581.21 ratings each month from 1995 to 2009. To this, we use cross-validation to find the optimal λ for the regularized movie + user + genre + year + month effects model. This data changes in average less than the above yearly data. To this, we run cross-validation again to find the optimal λ for the regularized movie + user + genre + year + month effects model.

```
rmses_cv5 <- sapply(lambdas, function(lam){
  reg_movie <- train_set %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(lam + n())) #reg_movie will contain bi

  reg_user <- train_set %>%
    left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
    group_by(userId) %>%
    summarize(bu = sum(rating - mu - bi)/(lam + n()))

  reg_genre <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - bi - bu)/(lam + n()))

  reg_year <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - bi - bu - b_g)/(lam + n()))

  reg_month <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    left_join(reg_year, by = "year") %>%
    group_by(month) %>%
    summarize(b_m = sum(rating - mu - bi - bu - b_g - b_y)/(lam + n()))

  predictions <- test_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    left_join(reg_year, by = "year") %>%
    left_join(reg_month, by = "month") %>%
    mutate(preds = mu + bi + bu + b_g + b_y + b_m) %>%
    pull(preds)

  return(RMSE(test_set$rating, predictions))
})
```

```
lambda <- lambdas[which.min(rmses_cv5)]
lambda
```

```
## [1] 4.75
```

The λ that minimizes the RMSE of the regularized movie + user + genre + year + month effects model is 4.75. Now we tune our model using the value of λ obtained.

```
reg_movie <- train_set %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(lambda + n())) #reg_movie will contain bi

reg_user <- train_set %>%
  left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
  group_by(userId) %>%
  summarize(bu = sum(rating - mu - bi)/(lambda + n()))

reg_genre <- train_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - bi - bu)/(lambda + n()))

reg_year <- train_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - bi - bu - b_g)/(lambda + n()))

reg_month <- train_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  left_join(reg_year, by = "year") %>%
  group_by(month) %>%
  summarize(b_m = sum(rating - mu - bi - bu - b_g - b_y)/(lambda + n()))

pred_8 <- test_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  left_join(reg_year, by = "year") %>%
  left_join(reg_month, by = "month") %>%
  mutate(preds = mu + bi + bu + b_g + b_y + b_m) %>%
  pull(preds)

rmse8 <- RMSE(test_set$rating, pred_8)
rmse8
```

```
## [1] 0.8655208
```

This model achieved a RMSE of 0.8655208.

Results

The following table compares the RMSE of all models used so far. Based on this, the model with the lowest RMSE will be selected to be used with the **validation** dataset.

Method	RMSE	Pred_index
Reg Movie/User/Genres/Year Effects	0.8655207	pred_7
Reg Movie/User/Genres/Year/Month Effects	0.8655208	pred_8
Reg Movie/User/Genres Effects	0.8655406	pred_6
Reg Movie/User Effects	0.8655424	pred_4
Movie + User Effects	0.8661625	pred_2
Ensemble of predictions 1,2,3,4(just the mean)	0.8847612	pred_5
Reg Movie Effects	0.9436775	pred_3
Movie Effects	0.9437144	pred_1

The model with lowest RMSE is the Regularized Movie + User + Genres + Year Effects, and therefore it will be used for validation purposes.

As mentioned above, for the validation we will use the *Regularized Movie + User + Genres + Year Effects* model with the validation set. First, we prepare the edx and validation sets by adding the **month** and **year** columns.

We will tune our model with a $\lambda = 4.75$, since this is the optimal value for this model obtained before.

```
edx <- edx %>%
  mutate(year = year(as_datetime(timestamp)),
         month = month(as_datetime(timestamp)))

validation <- validation %>%
  mutate(year = year(as_datetime(timestamp)),
         month = month(as_datetime(timestamp)))

lambda <- lambdas[which.min(rmses_cv4)]

reg_movie <- edx %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(lambda + n())) #reg_movie will contain bi

reg_user <- edx %>%
    left_join(reg_movie, by = "movieId") %>%
  #need to join so that we have access to bi
    group_by(userId) %>%
    summarize(bu = sum(rating - mu - bi)/(lambda + n()))

reg_genre <- edx %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - bi - bu)/(lambda + n()))

reg_year <- edx %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
```

```
    left_join(reg_genre, by = "genres") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - bi - bu - b_g)/(lambda + n()))

pred_final <- validation %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    left_join(reg_year, by = "year") %>%
    mutate(preds = mu + bi + bu + b_g + b_y) %>%
    pull(preds)

final_rmse <- RMSE(validation$rating, pred_final)
final_rmse
```

```
## [1] 0.8648005
```

Using this model, we obtain the lowest RMSE of 0.8648005. We can say that this model captures the noise in the training dataset and makes more accurate estimations by reducing the bias of the model, and therefore reducing the risk of overfitting.

Conclusion

We conclude that the most effective and accurate model, based on the specific characteristics of the dataset, is the model that Regularized the movie, user, genre, and year effects. Using this model we get a RMSE of 0.8648005 when applying it to the validation set.

As a potential extension of this model we can think on the possibility of providing recommendations to users. This requires finding movies that a user did not rate, filtering through the selected movies and choosing only ratings of either 4 or 5, and filtering by the user's most watched genres.