Movielens Recommendation System Report

Humberto Garza

25/Oct/2022

Contents

1 Introduction

The MovieLens data set contains a set of movie ratings from the MovieLens website, a movie recommendation service. This data set was collected and maintained by GroupLens, a research group at the University of Minnesota, it contains over 20 million ratings for over 27,000 movies by more than 138,000 users and its widely used in education, research, and industry and it is downloaded hundreds of thousands of times each year reflecting their use in popular press, programming books, traditional and online courses, and software

The goal of this project is to create a model that can predict ratings for unseen data, 90% of this data set is for training and 10% for testing. The rating metric for the movies is based on a scale of 0 to 5 stars. An RMSE of 0 means we are always correct, an RMSE of 1 means the predicted ratings are off by 1 star, to evaluate how successful our model is we will use the residual mean squared error (RMSE) which can be interpreted as the typical error between the predicted and actual value of a movie rating, we are looking to obtain an RMSE value <= 0.86490 on the test set.

This project is for Data Science: Capstone (PH125.9x) course in the HarvardX Professional Certificate Data Science Program

2 Analysis

2.1 Data Acquisition

We download, prepare and split the data with the code that was given to us in the course instructions.

```r
##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

# Note: this process could take a couple of minutes

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(dslabs)
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

dl <- tempfile()
download.file("https://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:
```

```r
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)`
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)
```

Per course instructions the edx dataset should be split into training and test sets, and use the validation data set only to test our final model to avoid over fitting.

```r
set.seed(9, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index, ]
test_set <- edx[test_index,]
```

Make sure we don’t include users and movies in the test set that do not appear in the training set.

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

2.2 Exploratory Data Analysis

The result of the previous code is two data set tables in tidy format with millions of rows, each row represents a rating given by one user to one movie, this data sets contain six columns: "userId", "movieId", "rating", "timestamp", "title", and "genres"

```
edx[1:6,] %>% knitr::kable()
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

We can see the amount of different users, movies and genres:

```
edx %>%
summarize(usersNum = n_distinct(userId),
moviesNum = n_distinct(movieId),
genresNum = n_distinct(genres)) %>%
knitr::kable()
```

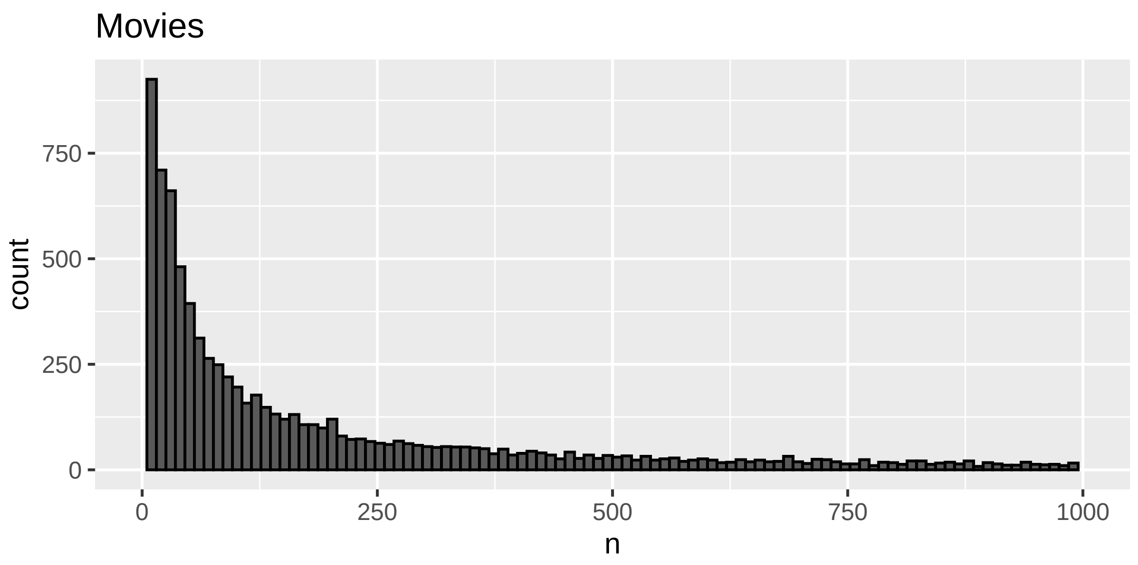
usersNum	moviesNum	genresNum
69878	10677	797

Check for the amount of missing values.

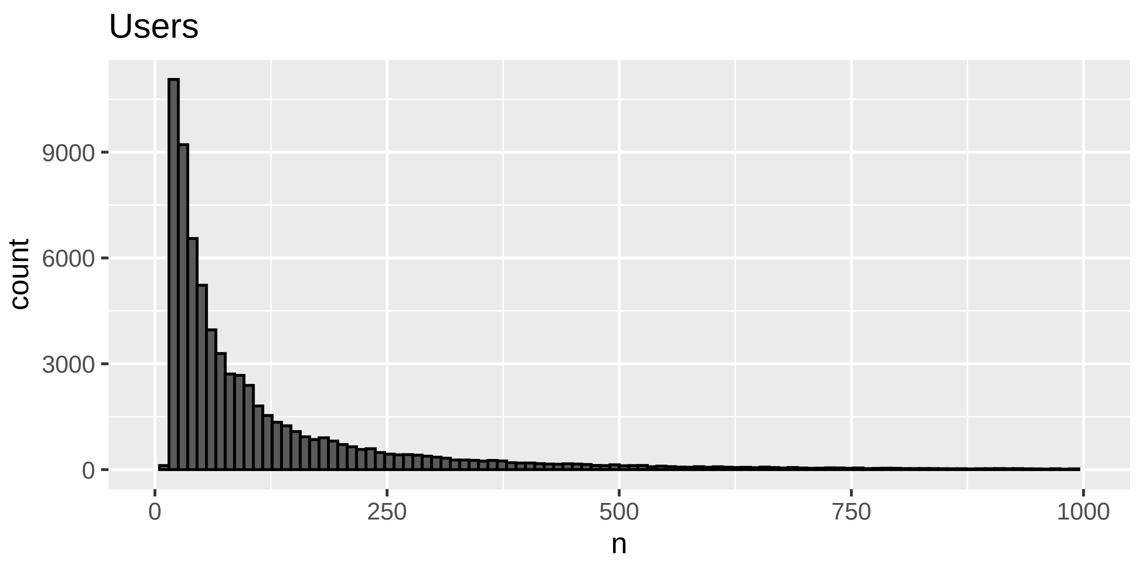
```
edx %>% summarize(userId = sum(is.na(userId)), movieId = sum(is.na(movieId)),
rating = sum(is.na(rating)), timestamp = sum(is.na(timestamp)),
title = sum(is.na(title)), genres = sum(is.na(genres))) %>%
knitr::kable()
```

userId	movieId	rating	timestamp	title	genres
0	0	0	0	0	0

There are no missing values in the data set. We will take a look at the distribution of movie ratings.

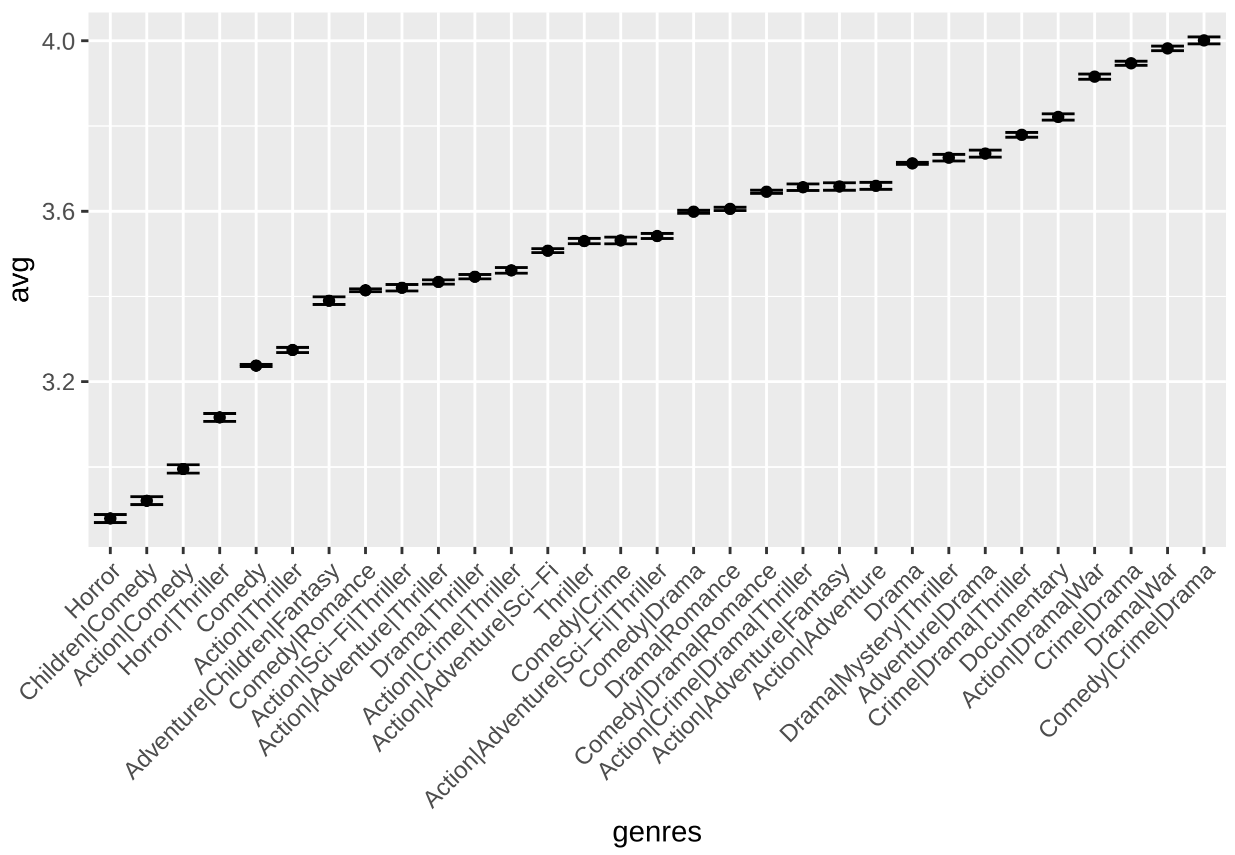


From the chart we can observe a huge imbalance in the number of ratings, some movies have a disproportionately large amount of ratings and others have next to none, that will be an important reason to consider the "movieId" variable in our model. Now lets take a look at the userId variable



Again we notice a similar situation, another variable to consider for the model, we can observe great variation in the ratings when we group by users ID, this may be explained by user preferences and taste.

Now lets take a look at the movie genres, since its not a numeric variable we group by unique genres combination, and count the amount of movies on each of them.



The movie genres clearly plays a role in the rating of the movie, with the "Drama" being the one contained in the highest rated genre combination and the ones that contain "Horror" and "comedy" being among the worst rated.

2.3 Loss function

The residual mean squared error (RMSE) is the square root of the average squared error. It is the typical metric to evaluate recommendation systems, it will be used as a metric to evaluate our models and it is defined as:

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

with $y_{u,i}$ representing the rating for movie i by user u and $\hat{y}_{u,i}$ as our prediction, N as the number of user/movie combinations and the sum of all the combinations.

This is how the RMSE function looks on code:

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

3 Models

3.1 First model (naive aproach)

The simplest approach at building a recommender system is assuming the same rating for all movies without even considering the differences in user preferences, movie title or the different genres

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

with μ the average of all movie ratings and $\varepsilon_{i,u}$ as independent errors explained by random variation

```
# Calculate the average of all movie ratings
avg <- mean(train_set$rating)
# Apply RMSE function and saving result
naive <- RMSE(test_set$rating, avg)
# Add model name and result to a table
models <- tibble(Method = "Just the average", RMSE = naive)
```

We can observe the RMSE value obtained in this table we created.

```
models %>% knitr::kable()
```

Method	RMSE
Just the average	1.060091

3.2 Modeling Movie Effects

Naturally some movies will have a higher rating than others, and the data confirms this, so in order to improve our model we add the term b_i which represents the average ranking for movie i also known as "bias", in other words we group movies by its ID and calculate the mean value of each

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

```
# Obtaining the b_i term
movie_avgs <- train_set %>% group_by(movieId) %>% summarize(b_i = mean(rating - avg))
```

```
# Calculating predictions vector
predictions <- test_set %>% left_join(movie_avgs, by = 'movieId') %>%
mutate(pred = b_i + avg) %>% .$pred
# Apply RMSE function and saving result
model_1_rmse <- RMSE(test_set$rating, predictions)
# Calculate imporvement percentage

# Add model name and result to a table
models <- bind_rows(models, tibble(Method = "Movie Effects", RMSE = model_1_rmse))
```

here we can see a 10.98% improvement

```
models %>% knitr::kable()
```

Method	RMSE
Just the average	1.0600912
Movie Effects	0.9437409

3.3 Modeling Movie Effects + User Effects

We will try to improve the accuracy of our model by adding another term, the bias of a given user b_u and calculate how much it helps, the new formula would look like this:

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

```
# Obtaining the b_i term
user_avgs <- train_set %>% left_join(movie_avgs, by='movieId') %>%
group_by(userId) %>% summarize(b_u = mean(rating - avg - b_i))
# Making predictions
predictions <- test_set %>% left_join(movie_avgs, by='movieId') %>%
left_join(user_avgs, by='userId') %>% mutate(pred = avg + b_i + b_u) %>%
.$pred
# Apply RMSE function and saving result
model_2_rmse <- RMSE(test_set$rating, predictions)
# Add model name and result to a table
models <- bind_rows(models, tibble(Method = "Movie Effects + User Effects",
RMSE = model_2_rmse))
```

```
models %>% knitr::kable()
```

Method	RMSE
Just the average	1.0600912
Movie Effects	0.9437409
Movie Effects + User Effects	0.8658039

Another improvement, 8.26% with this model.

3.4 Modeling Movie Effects + User Effects + Genres Effects

As we saw in the data exploration section, the genre of the movie has an effect on the movie rating, so we will add the genre effect to the model and explore if there is any improvement, the equation will look as follows:

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

```
# Obtaining the genres bias effect "b_g"
genres_avgs <- train_set %>% left_join(movie_avgs, by = 'movieId') %>%
left_join(user_avgs, by = 'userId') %>% group_by(genres) %>%
summarize(b_g = mean(rating - avg - b_i - b_u))
# Calculating predictions vector
predictions <- test_set %>% left_join(movie_avgs, by = 'movieId') %>%
left_join(user_avgs, by = 'userId') %>%
left_join(genres_avgs, by = 'genres') %>%
mutate(pred = avg + b_i + b_u + b_g) %>% .$pred
# Apply RMSE function and saving results
model_3_rmse <- RMSE(test_set$rating, predictions)
# Add model name and result to a table
models <- bind_rows(models, tibble(Method = "Movie + User + Generes Effect",
RMSE = model_3_rmse))
```

We can observe a negligible improvement. we have been hit by the law of diminishing returns apparently

```
models %>% knitr::kable()
```

Method	RMSE
Just the average	1.0600912
Movie Effects	0.9437409
Movie Effects + User Effects	0.8658039
Movie + User + Generes Effect	0.8654813

When we take a look at the top 10 best and worst rated movies based on the movie bias (b_i) parameter we calculated, we can see that they have very few ratings, this is the top 10 best movies and the amount of ratings they have

```
# Make a data frame with only movieId and title
movie_titles <- train_set %>% select(movieId, title) %>% distinct ()

train_set %>% count(movieId) %>% left_join(movie_avgs) %>%
left_join(movie_titles, by="movieId") %>% arrange(desc(b_i)) %>%
select(title, b_i, n) %>% .[1:10,] %>% knitr::kable()
```

title	b_i	n
Hellhounds on My Trail (1999)	1.487539	1
Satan's Tango (Sátántangó) (1994)	1.487539	2
Shadows of Forgotten Ancestors (1964)	1.487539	1
Sun Alley (Sonnenallee) (1999)	1.487539	1
Class, The (Entre les Murs) (2008)	1.487539	2
Blue Light, The (Das Blaue Licht) (1932)	1.487539	1
Hospital (1970)	1.487539	1
Human Condition III, The (Ningen no joken III) (1961)	1.320872	3
Titicut Follies (1967)	1.237539	2
Constantine's Sword (2007)	1.237539	2

And the 10 worst:

```
train_set %>% count(movieId) %>% left_join(movie_avgs) %>%
left_join(movie_titles, by="movieId") %>% arrange(b_i) %>%
select(title, b_i, n) %>% .[1:10,] %>% knitr::kable()
```

title	b_i	n
30 Years to Life (2001)	-3.012461	1
Besotted (2001)	-3.012461	2
Hi-Line, The (1999)	-3.012461	1
Grief (1993)	-3.012461	1
Accused (Anklaget) (2005)	-3.012461	1
War of the Worlds 2: The Next Wave (2008)	-3.012461	2
SuperBabies: Baby Geniuses 2 (2004)	-2.679128	45
Hip Hop Witch, Da (2000)	-2.666307	13
Disaster Movie (2008)	-2.608615	26
From Justin to Kelly (2003)	-2.606974	164

To try to correct this we will use regularization which permits us to penalize large estimates that are formed using small sample sizes.

3.5 Choosing The Penalty Term For Regularized Movie Effect

The general idea behind the regularization process is to limit the total variability of the effect sizes, to avoid a small amount of ratings having a big impact on the size of the bias. Here is what the formula looks like when we add the penalized least squares feature to it.

$$\sum_{u,i} (y_{u,i} - \mu - b_i)^2 + \lambda \sum_i b_i^2$$

Based on this, the equation that minimizes the values of b_i will look like this:

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

when our sample size n_i is very large the penalty λ has little impact on the estimate since $n_i + \lambda \approx n_i$ and, when the sample size is small (n_i), the impact of λ increases, shrinking the estimate $\hat{b}_i(\lambda)$ towards zero

First we use cross-validation to choose the tuning parameter lambda (λ)

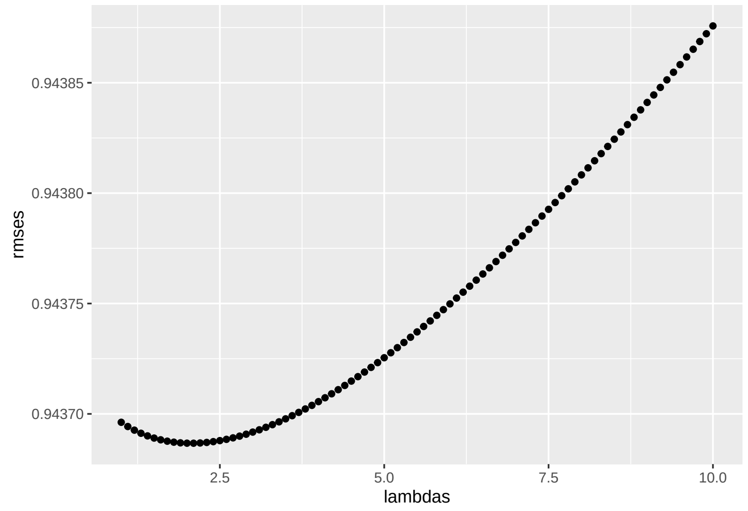
```r
# Make a vector with a range of values for lambda with increments of 0.1
lambdas <- seq(1, 10, 0.1)

# Making dataframe with half of the equation in the variable "s"
#and the number of ratings per movie n_i
just_the_sum <- train_set %>% group_by(movieId) %>%
summarize(s = sum(rating - avg), n_i = n())
# Funtion to run vector lambdas to calculate optimal value
rmses <- sapply(lambdas, function(l){
# Calculating regularized b_i and saving predictions vector
predicted_ratings <- test_set %>% left_join(just_the_sum, by='movieId') %>%
mutate(b_i = s/(n_i+l)) %>% mutate(pred = avg + b_i) %>% .$pred
# Apply RMSE function and returning results
return(RMSE(predicted_ratings, test_set$rating))
})

# Add model name and result to a table
models <- bind_rows(models, tibble(Method = "Regularized Movie Effect",
RMSE = min(rmses)))

# Save the best tunning parameter in a variable
lambda <- lambdas[which.min(rmses)]
```

```
# We plot the lambdas vector against the results of the rmses calculations
qplot(lambdas, rmses)
```



```
models %>% knitr::kable()
```

Method	RMSE
Just the average	1.0600912
Movie Effects	0.9437409
Movie Effects + User Effects	0.8658039
Movie + User + Generes Effect	0.8654813
Regularized Movie Effect	0.9436867

3.6 Choosing The Penalty Term For Regularized Movie + User Effects

We can use regularization for the estimate Regularized Movie + User Effects as well, the equation will look like this

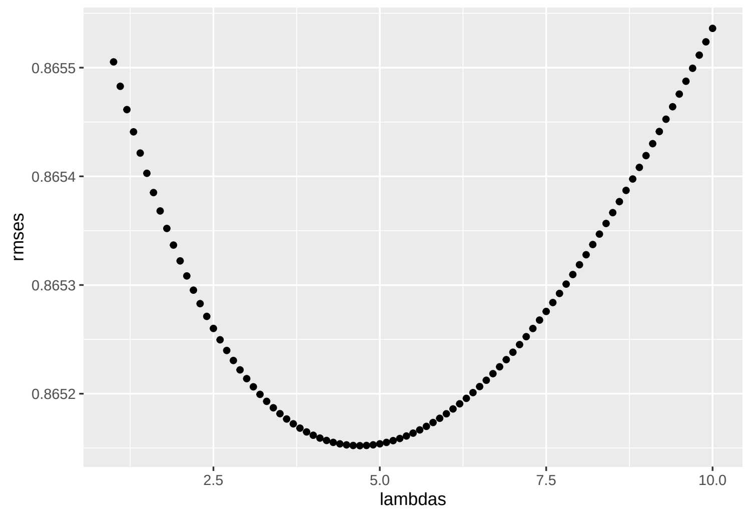
$$\sum_{u,i} (y_{u,i} - \mu - b_i - b_u)^2 + \lambda \left(\sum_i b_i^2 + \sum_u b_u^2 \right)$$

```
# Make a vector with a range of values for lambda with increments of 0.1
lambdas <- seq(1, 10, 0.1)
# Funtion to run vector lambdas to calculate optimal value
rmses <- sapply(lambdas, function(l){
# Calculating regularized b_i vector
b_i <- train_set %>% group_by(movieId) %>% summarize(b_i = sum(rating - avg)/(n()+l))
# Calculating regularized b_u vector
b_u <- train_set %>% left_join(b_i, by="movieId") %>% group_by(userId) %>%
summarize(b_u = sum(rating - b_i - avg)/(n()+1))
# Making predictions
predictions <- test_set %>% left_join(b_i, by = "movieId") %>%
left_join(b_u, by = "userId") %>% mutate(pred = avg + b_i + b_u) %>% .$pred
# Apply RMSE function and returning results
return(RMSE(test_set$rating, predictions))
})

# Save the best tunning parameter in a variable
lambda <- lambdas[which.min(rmses)]

# Add model name and result to a table
models <- bind_rows(models, tibble(Method="Regularized Movie + User Effect Model",
RMSE = min(rmses)))
```

```
# We plot the lambdas vector against the results of the rmses calculations
qplot(lambdas, rmses)
```



Method	RMSE
Just the average	1.0600912
Movie Effects	0.9437409
Movie Effects + User Effects	0.8658039
Movie + User + Generes Effect	0.8654813
Regularized Movie Effect	0.9436867
Regularized Movie + User Effect Model	0.8651522

3.7 Choosing The Penalty Term For Regularized Movie + User + Genres Effects

We can add a genres term to this regularized model, the estimate Regularized Movie + User + Genres Effects equation will look like this:

$$\sum_{u,i}(y_{u,i}-\mu-b_i-b_u-b_g)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2+\sum_g b_g^2\right)$$

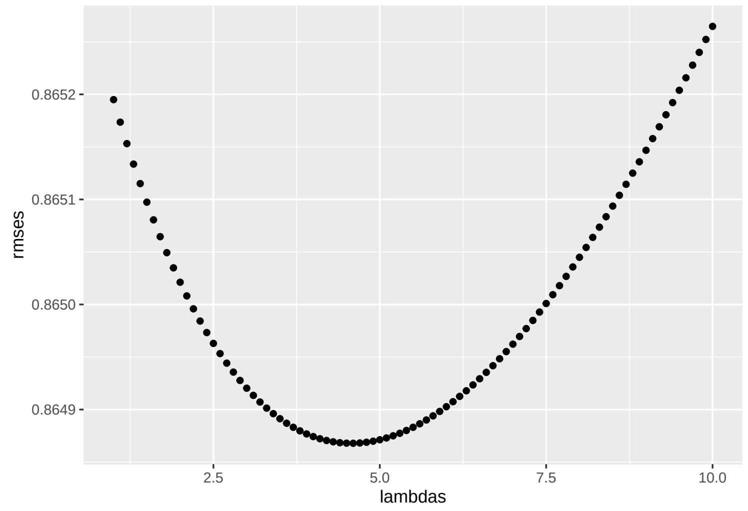
```r
# Make a vector with a range of values for lambda with increments of 0.1
lambdas <- seq(1, 10, 0.1)
# Funtion to run vector lambdas to calculate optimal value
rmses <- sapply(lambdas, function(l){
# Calculating regularized b_i vector
b_i <- train_set %>% group_by(movieId) %>% summarize(b_i = sum(rating - avg)/(n()+l))
# Calculating regularized b_u vector
b_u <- train_set %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
summarize(b_u = sum(rating - b_i - avg)/(n()+l))
# Calculating regularized b_g vector
b_g <- train_set %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>%
group_by(genres) %>% summarize(b_g = sum(rating - b_i - b_u - avg)/(n()+l))


# Making predictions
predictions <- test_set %>% left_join(b_i, by = "movieId") %>%
left_join(b_u, by = "userId") %>% left_join(b_g, by = "genres") %>%
mutate(pred = avg + b_i + b_u + b_g) %>% .$pred
# Apply RMSE function and returning results
return(RMSE(test_set$rating, predictions))
})

# Save the best tuning parameter in a variable
lambda <- lambdas[which.min(rmses)]

# Add model name and result to a table
models <- bind_rows(models, tibble(Method="Regularized Movie + User + Genres Effect Model",
RMSE = min(rmses)))
```

```r
# We plot the lambdas vector against the results of the rmses calculations
qplot(lambdas, rmses)
```



Method	RMSE
Just the average	1.0600912
Movie Effects	0.9437409
Movie Effects + User Effects	0.8658039
Movie + User + Generes Effect	0.8654813
Regularized Movie Effect	0.9436867
Regularized Movie + User Effect Model	0.8651522
Regularized Movie + User + Genres Effect Model	0.8648679

The goal of RMSE < 0.86490 has been reached, lets try this model with our tuning parameter on the validation test and check if it holds out.

```
# Using complete edx data set to obtain average of all movie ratings
avg  <- mean(edx$rating)
# Calculating edx regularized b_i data frame
b_i <- edx %>% group_by(movieId) %>% summarize(b_i = sum(rating - avg)/(n()+lambda))
# Calculating edx regularized b_u data frame
b_u <- edx %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
summarize(b_u = sum(rating - b_i - avg)/(n()+lambda))
# Calculating edx regularized b_g data frame
b_g <- edx %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>%
group_by(genres) %>% summarize(b_g = sum(rating - b_i - b_u - avg)/(n()+lambda))

# Making predictions with final validation test set
predictions <- validation %>% left_join(b_i, by = "movieId") %>%
left_join(b_u, by = "userId") %>% left_join(b_g, by = "genres") %>%
mutate(pred = avg + b_i + b_u + b_g) %>% .$pred

# Save RMSE result in variable
final_model <- RMSE(validation$rating, predictions)

# Add model name and result to a table
models <- bind_rows(models, tibble(Method = "Final Model", RMSE = final_model))
```

```
final_model
```

```
## [1] 0.8644529
```

4 Results

In the table below we display the RMSE values of our models and we can appreciate the gradual improvements obtained with each factor added to the model.

```
models %>% knitr::kable()
```

Method	RMSE
Just the average	1.0600912
Movie Effects	0.9437409
Movie Effects + User Effects	0.8658039
Movie + User + Generes Effect	0.8654813
Regularized Movie Effect	0.9436867
Regularized Movie + User Effect Model	0.8651522
Regularized Movie + User + Genres Effect Model	0.8648679
Final Model	0.8644529

5 Conclusion

We obtained an RMSE value of 0.8644529, which confirms the model is successful making predictions within the error range that was given to us in the course instructions, for now we will stop trying to improve and refine this model, since every factor we add increases our computational load and the calculations time, other predictive algorithms have proven to be impractical for our computing constrains and for a data set this big.

With data sets of this size, calculating the optimal lambda values for this regularized models is getting too computationally intensive for a laptop, in the future it will be interesting to explore other more advanced predictive methods such as “Gradient Descent” that is widely used in the field and “Slope One algorithm” which is way more successful than our model at making predictions for recommendation systems