Movielens Machine Learning Project Report

Lorenzo Luna

06/12/2019

Intro

We will be implementing a recommendation system following the approach explained in this course.

The movielens dataset is made up of 10 million observations, each corresponding to a rating given to a movie by an user. Each observation has 6 variables:

```
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

The variables userId and movieId are numeric identifiers for users and movies respectively. The rating variable is the numeric rating given by the indicated user to the indicated movie, in a scale from 0.5 to 5 by increments of 0.5 (due to half star ratings). The variables title and genre are characters and indicate the title of the movie (which may not be unique, therefore it is important to distinguish movies by movieId) and the genres associated with it. The timestamp date variable indicates when the rating was expressed.

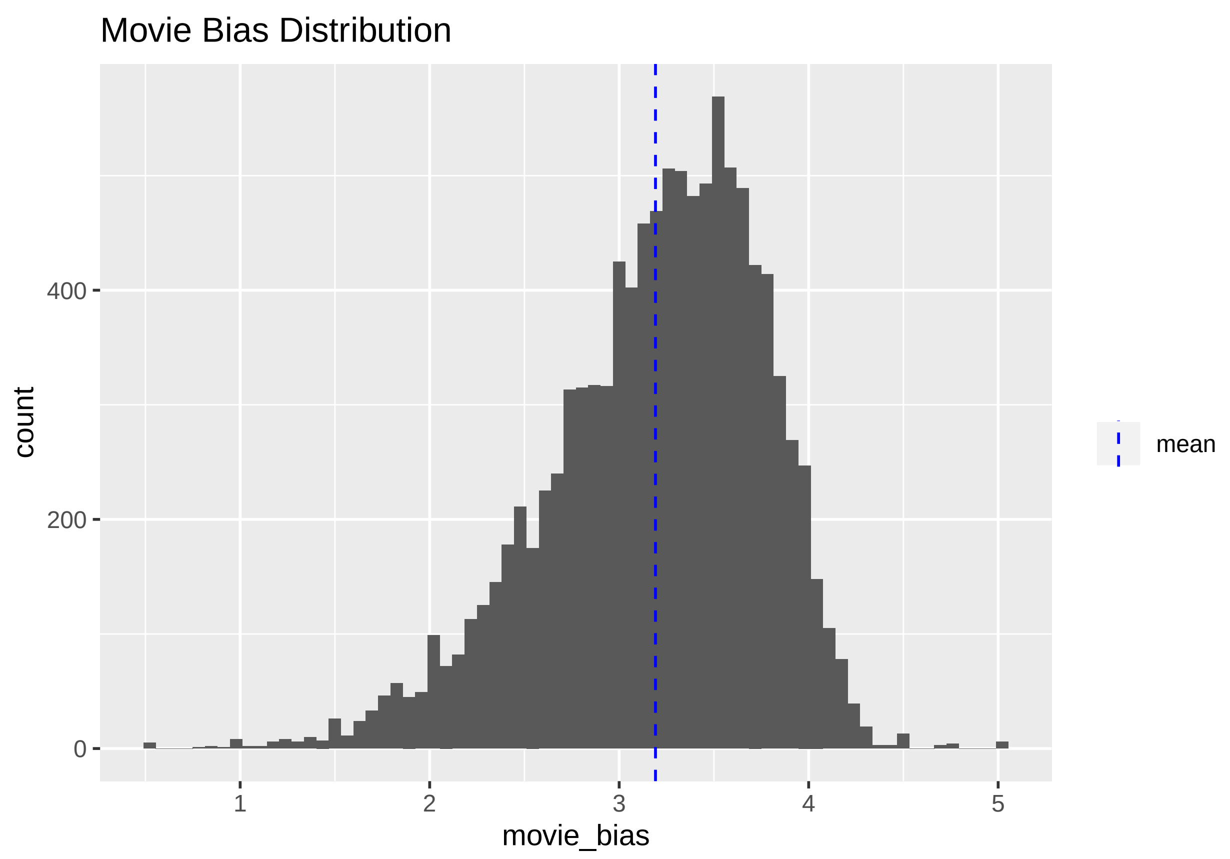
Predicting a user's rating of a movie is a regression machine learning problem because we are predicting a numerical value. We will use a linear model that predicts ratings accounting for the biases of specific movies and users, caused by movies being either good or bad, and users having higher or lower quality standards.

To increase the precision of our model, we will regularize these biases to penalize users and movies for which we have too little data to make a well-informed prediction, tuning the λ parameter with cross-validation.

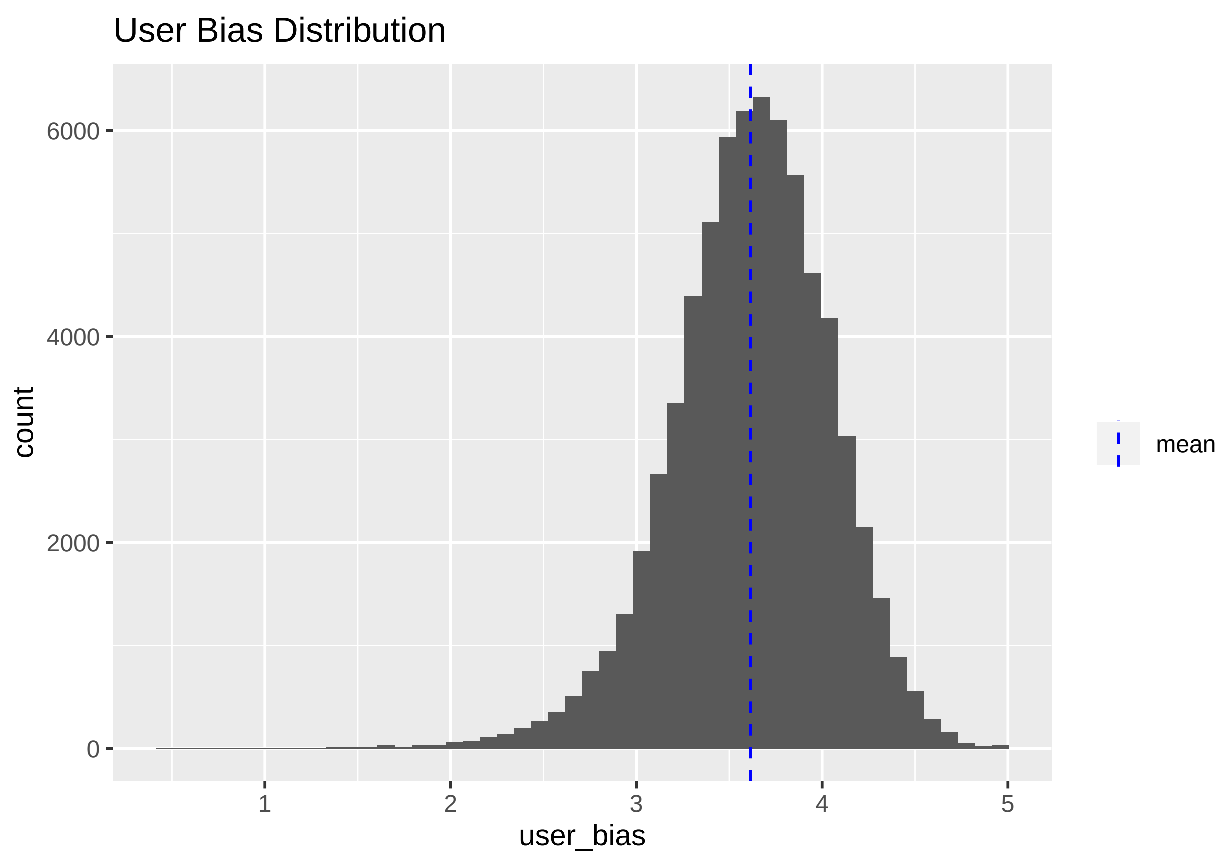
Method

The data shows that there is a bias effect caused by both the users and the movies, thus we will be estimating this effect for each user and movie, in order to predict ratings expressed by those users on movies they previously didn't rate.

```
edx %>%
  group_by(movieId) %>%
  summarize(movie_bias = mean(rating)) %>%
  ggplot(aes(movie_bias)) +
  geom_histogram(bins = 70) +
  geom_vline(aes(xintercept = mean(movie_bias), color = "mean"),
             linetype = "dashed", size = .5, show.legend = TRUE) +
  scale_color_manual(name = "", values = c("blue")) +
  ggtitle("Movie Bias Distribution")
```

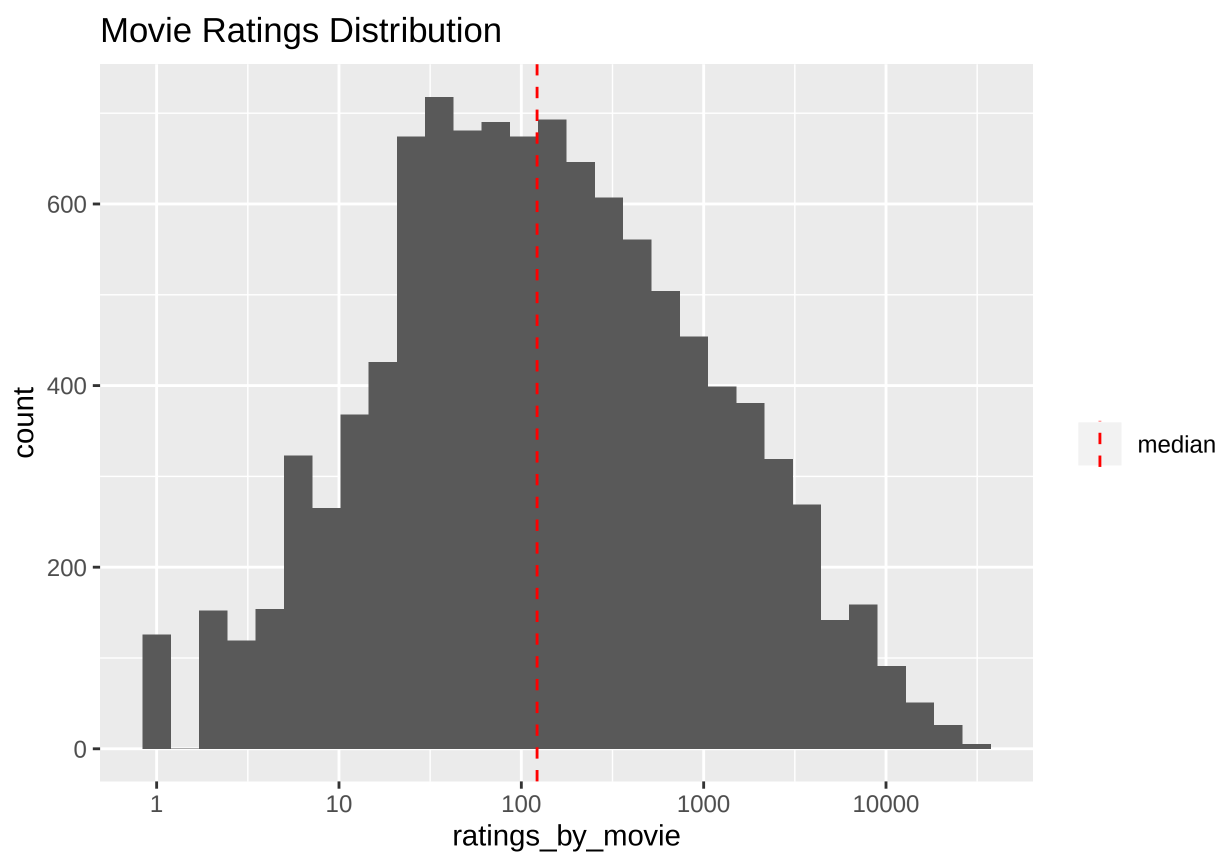


```
edx %>%
  group_by(userId) %>%
  summarize(user_bias = mean(rating)) %>%
  ggplot(aes(user_bias)) +
  geom_histogram(bins = 50) +
  geom_vline(aes(xintercept = mean(user_bias), color = "mean"),
             linetype = "dashed", size = .5, show.legend = TRUE) +
  scale_color_manual(name = "", values = c("blue")) +
  ggtitle("User Bias Distribution")
```

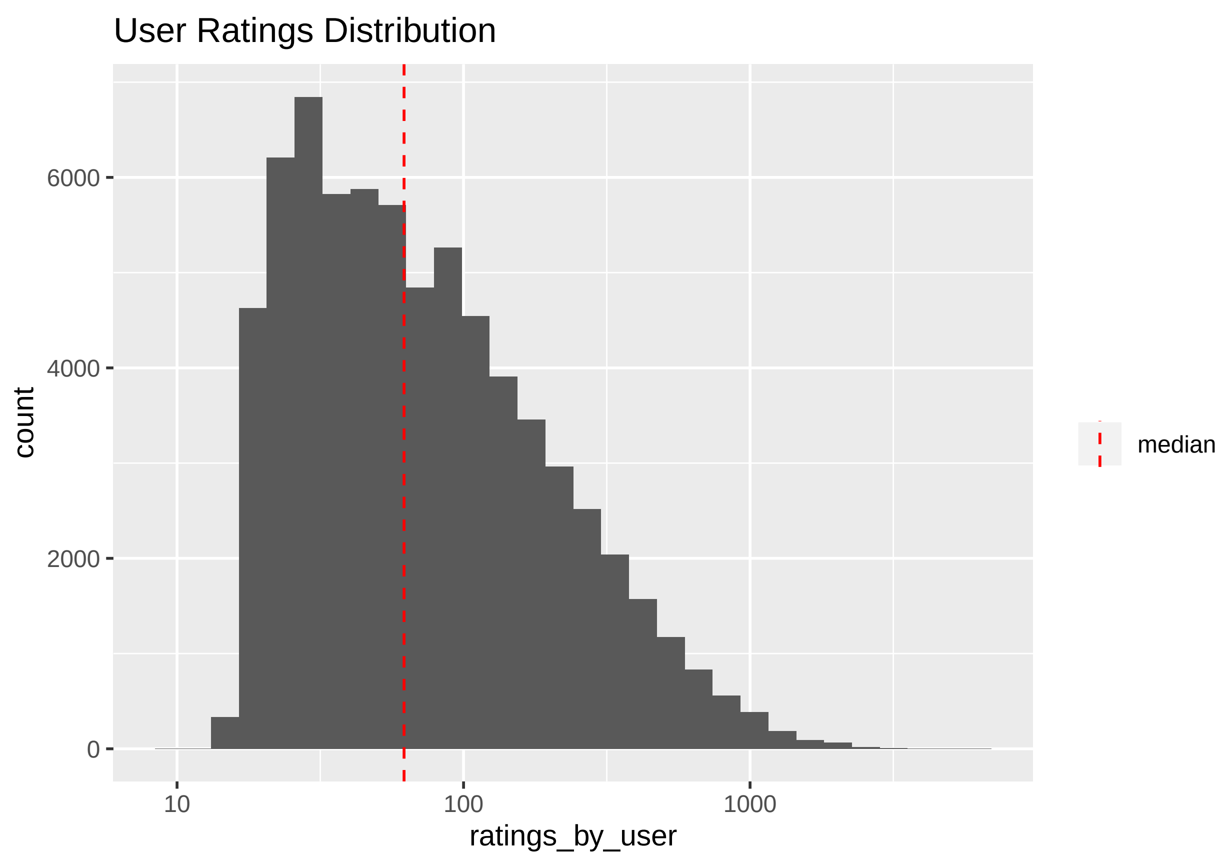


As previously shown during the course, the number of movies that have only received few ratings is very high with half the movies having received less than about 125 ratings. Because of this regularization might help with improving the accuracy of the model. We also see that similarly, most users have expressed less than about 65 ratings, with a steadily decreasing amount of users being more prolific in expressing ratings. Regularization can be helpful here too.

```
edx %>%
  group_by(movieId) %>%
  summarize(ratings_by_movie = n()) %>%
  ggplot(aes(x = ratings_by_movie, y = ..count..)) +
  geom_histogram(bins = 30) +
  geom_vline(aes(xintercept = median(ratings_by_movie), color = "median"),
             linetype = "dashed", size = .5, show.legend = TRUE) +
  scale_x_log10() +
  scale_color_manual(name = "", values = c("red")) +
  ggtitle("Movie Ratings Distribution")
```



```r
edx %>%
  group_by(userId) %>%
  summarize(ratings_by_user = n()) %>%
  ggplot(aes(x = ratings_by_user, y = ..count..)) +
  geom_histogram(bins = 30) +
  geom_vline(aes(xintercept = median(ratings_by_user), color = "median"),
             linetype = "dashed", size = .5, show.legend = TRUE) +
  scale_x_log10() +
  scale_color_manual(name = "", values = c("red")) +
  ggtitle("User Ratings Distribution")
```



Given these observation, we will train a simple machine learning algorithm that estimates regularized biases for each user and movie and uses those, as well as the overall mean of ratings, to make a prediction on unknown ratings, through the following model:

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

Where $Y_{u,i}$ is the rating from user u of movie i, $\varepsilon_{u,i}$ is an indipendent and identically distributed random variable representing the error in our estimate, μ is the mean of all ratings, b_i is the regularized bias effect for movie i and b_u is the regularized bias effect for user u. These biases are regularized using the optimal parameter λ determined using cross-validation on the test set. They are estimated as follows:

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

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

The goal is to minimize the root mean squared error (RMSE) of our prediction, defined as such:

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

Where $\hat{y}_{u,i}$ is our prediction of the true value $y_{u,i}$.

Results

After running the provided initial setup code we split the edx dataset into a training and a test set, in order to avoid using the validation dataset as a test set

```r
#create training and test sets from edx

train_ind = createDataPartition(edx$rating, p = 0.9, list = FALSE)
train_edx = edx[train_ind,]
test_edx = edx[-train_ind,]

#remove from test set users and movies not present in training set

test_edx <- test_edx %>%
  semi_join(train_edx, by = "movieId") %>%
  semi_join(train_edx, by = "userId")
```

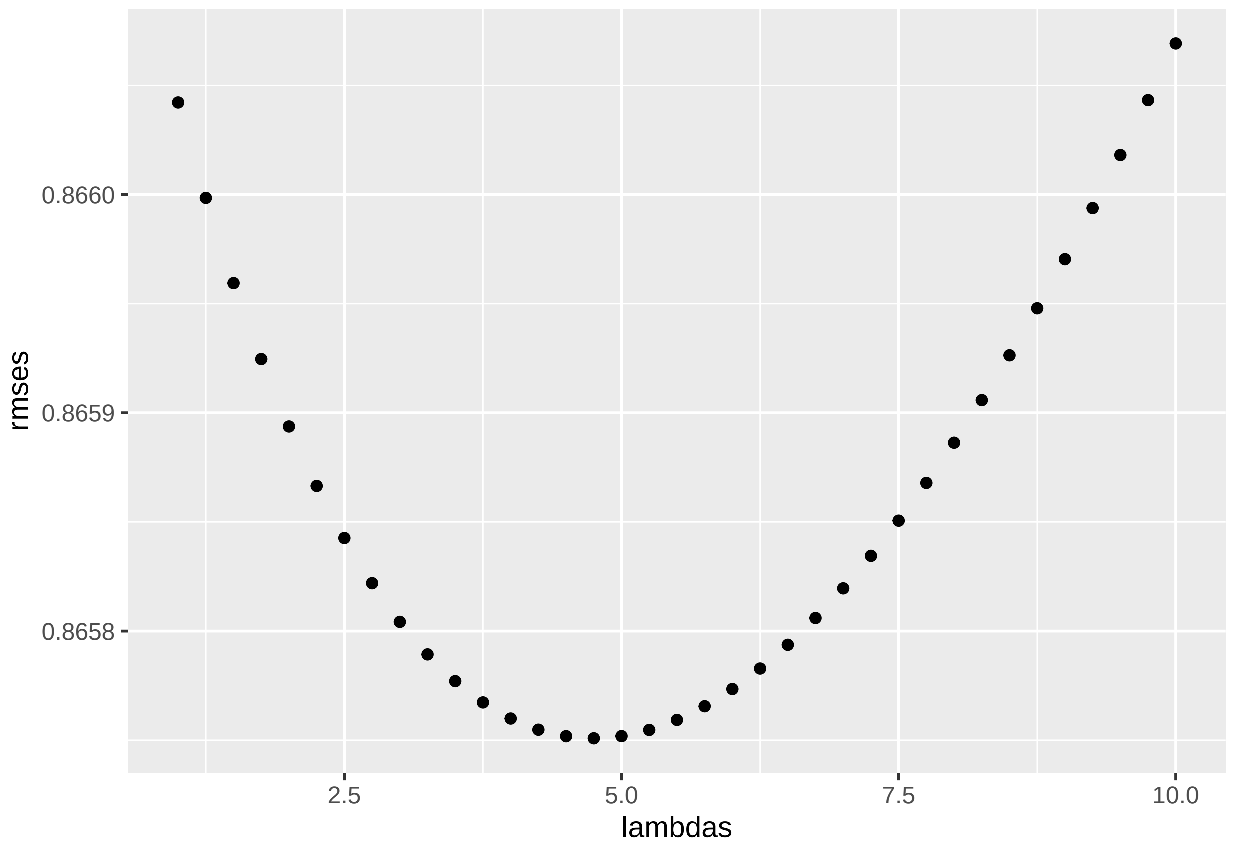
We define the RMSE function, that calculates the root mean squared error of our estimate. We will use this function to evaluate the accuracy of our prediction.

```r
#define RMSE output function for solution evaluation

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

We now want to optimize the tuning parameter λ by using cross-validation on the test set we've created.

```r
#optimize tuning parameter lambda validating on the test set

lambda_RMSE = function(lambda) {
#calculate overall rating mean
mu <- mean(train_edx$rating)

#calculate regularized movie bias for each movie using lambda
b_i <- train_edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

#calculate regularized user bias using lambda
b_u <- train_edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

#predict ratings on the test set
predicted_ratings <-
  test_edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

return(RMSE(test_edx$rating, predicted_ratings))

}
#try lambdas from 1 to 10 by increments of 0.25
lambdas = seq(1,10,0.25)
rmses = sapply(lambdas, lambda_RMSE)
```

```r
#pick best lambda
lambda = lambdas[which.min(rmses)]

#show lambda and qplot
qplot(lambdas, rmses)
```

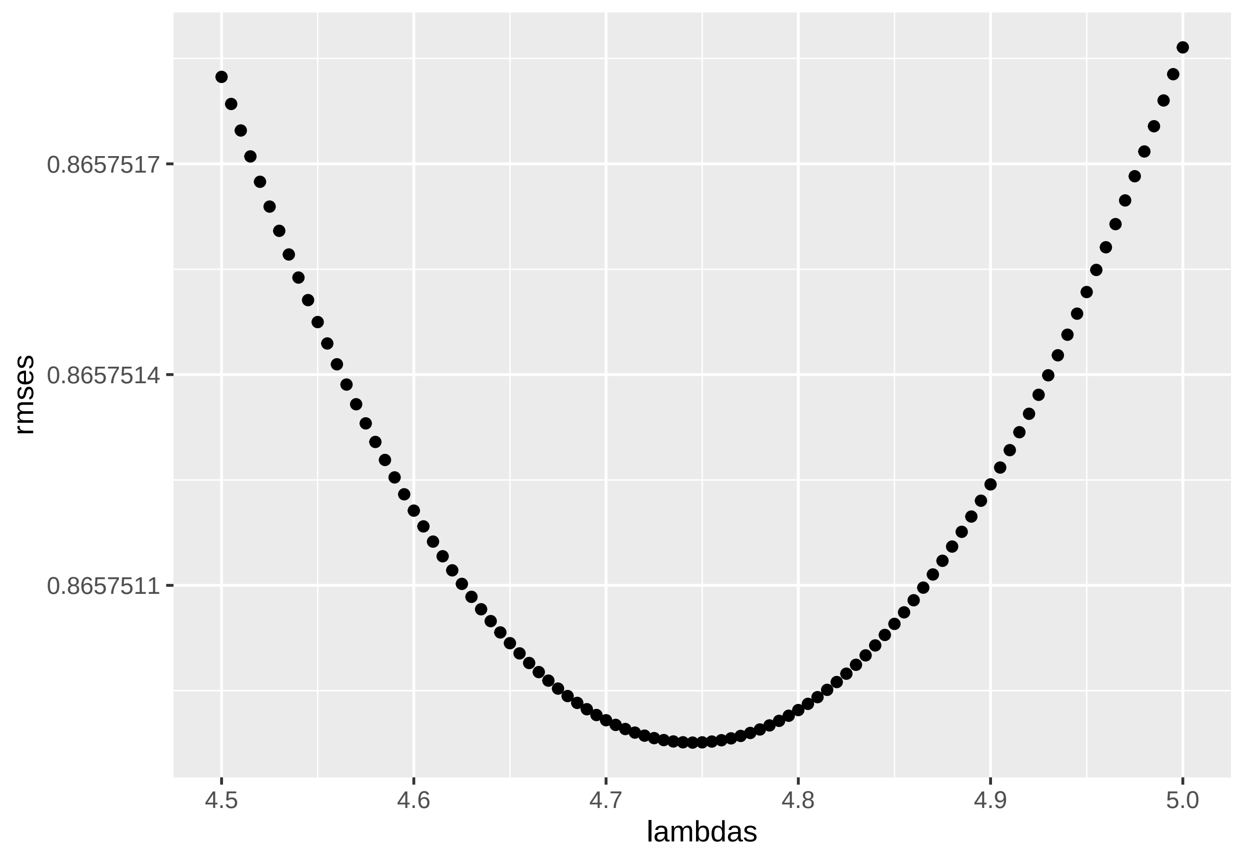


```r
lambda
```

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

As shown in the plot the optimal value of λ seems to be somewhere within a 0.25-neighborhood of the displayed value. We can further improve the precision of our optimization by testing a greater amount of values but within the neighborhood of the value.

```r
#try lambdas in the neighborhood of the previous optimum by increments of 0.005
lambdas = seq(lambda-0.25,lambda+0.25, 0.005)
rmses = sapply(lambdas, lambda_RMSE)

#pick best lambda
lambda = lambdas[which.min(rmses)]

#show lambda and qplot
qplot(lambdas, rmses)
```



```
lambda
```

```
## [1] 4.745
```

As shown by the console output the optimal value of λ has improved in precision compared to the earlier estimate.

We can now run the model on the validation set using the optimal estimate of the parameter λ and calculate the RMSE to evaluate its precision.

```
#calculate overall rating mean
mu <- mean(edx$rating)

#calculate regularized movie bias with optimized lambda
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

#calculate regularized user bias with optimized lambda
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

#predict ratings on the validation set
predicted_ratings <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

#display resulting RMSE
RMSE(validation$rating, predicted_ratings)
```

```
## [1] 0.8648202
```

We achieve an RMSE value lower than 0.8649.

Conclusion

We have defined a model for predicting a user's rating of a previously unrated movie based on the estimated user and movie biases. This captures the fact that users tend to have different (biased) quality standards when rating movies and the fact that movies tend to be rated depending on their quality, which corresponds to a deviation from the mean (a bias) in their ratings.

Regularizing the user and movie biases improves the accuracy of the model as the relevance of data with small sample sizes is reduced, since we have seen that this happens often in this dataset.

The model's accuracy is acceptable considering its computational simplicity which results in minimal run-time. This model is limited because it ignores the genre and timestamp variables, and doesn't model patterns in the data like users only liking a particular set of genres, or users liking a movie more if they liked movies in the same trilogy or series.

Such patterns would be better recognized by a machine learning algorithm utilizing clustering or matrix factorization on this dataset. This would likely require some form of dimensionality reduction to render the problem tractable, like Principal Component Analysis (PCA).