Report on MovieLens Project

Benedikt Bosch

26th July 2020

Overview over the MovieLens project

For this capstone project within Harvard's Professional Certificate in Data Science, I created a movie recommendation system using the MovieLens dataset and all the tools shown throughout the courses in this certificate. Within this project, I trained a machine learning algorithm using a training subset of the MovieLens dataset in order to predict the movie ratings in the complementary validation set. Doing so, my machine learning algorithm was able to yield a **RMSE** (residual mean squared error) of **0.86413** by using a **regularized movie + user + time + genre + publish effect model**.

MovieLens dataset

The original MovieLens dataset was generated by the GroupLens research lab. It includes over 20 million ratings for over 27,000 movies by more than 138,000 users. Within this project, I used a subset of the MovieLens dataset with 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. This dataset will be referred to in the following as MovieLens datset or as original datset. It can be found under the following link: https://grouplens.org/datasets/movielens/10m/

Approach and analysis overview

I developed my algorithm using the edx test set (90% of the data of the MovieLens dataset). For the final test of the algorithm, I predicted movie ratings in the validation set (10% of the data of the MovieLens dataset) as if they were unknown. To evaluate how close my predictions were to the true values in the validation set, RMSE (residual mean squared error) was used as loss function.

$$\text{RMSE} = \frac{\sqrt{\sum_{t=1}^{T}(\hat{y_t} - y_t)^2}}{T}$$

Within this function, $\hat{y_t}$ represents the prediction for each rating t and y_t the true value of the respective rating within the validation set.

Data cleaning and creation of train and validation sets

In a first step, after downloading the raw data, I inspected the head of the movie raw data (without ratings).

```r
kable(movies[1:6, ])
```

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action|Crime|Thriller

The excerpt of the raw data revealed that the data was not tidy as the movie title and the year the movie was published are concatenated. It was thus necessary to split the column values into two seperate pieces of information. Even if some movies from different years had the same name, they could still be uniquely identified by the movieId.

After splitting the movie title and the year of publication, I combined the movie raw data with the user ratings to create the MovieLens dataset. The head of the dataset looks as following.

Remark: *The year column is in character format and will be converted to date format for visualization purposes at a later stage.*

```r
kable(movielens[1:6, ])
```

userId	movieId	rating	timestamp	title	year	genres
1	122	5	838985046	Boomerang	1992	Comedy|Romance
1	185	5	838983525	Net, The	1995	Action|Crime|Thriller
1	231	5	838983392	Dumber	1994	Comedy
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	Generations	1994	Action|Adventure|Drama|Sci-Fi

As the timestamp column was not readable, it needed to be converted into the ISO 8601 format. For calculation purposes, this new date column is rounded by week.

Another look at the head of the cleaned MovieLens dataset reveals that the data is tidy now.

```r
kable(movielens[1:6, ])
```

userId	movieId	rating	title	year	genres	date
1	122	5	Boomerang	1992	Comedy|Romance	1996-08-04
1	185	5	Net, The	1995	Action|Crime|Thriller	1996-08-04
1	231	5	Dumber	1994	Comedy	1996-08-04
1	292	5	Outbreak	1995	Action|Drama|Sci-Fi|Thriller	1996-08-04
1	316	5	Stargate	1994	Action|Adventure|Sci-Fi	1996-08-04
1	329	5	Generations	1994	Action|Adventure|Drama|Sci-Fi	1996-08-04

Before exploring the data in more detail and training the machine learning algorithm, a train and test set needed to be created from the MovieLens data. The validation set contained 10% of the MovieLens data and was left untouched until the model was finalized. It was only used to compute the final RMSE. The train set (called edx set) was split into a "real" train set (90% of the edx set) and a test set to assess how close my rating predictions with different models were to the true values.

After cleaning the data and creating the train, test and validation set, the three datsets comprised the follwing number of ratings:

Dataset	Ratings
Train set	8100048
Test set	899988
Validation set	999999

Insights gained through data exploration and visualization

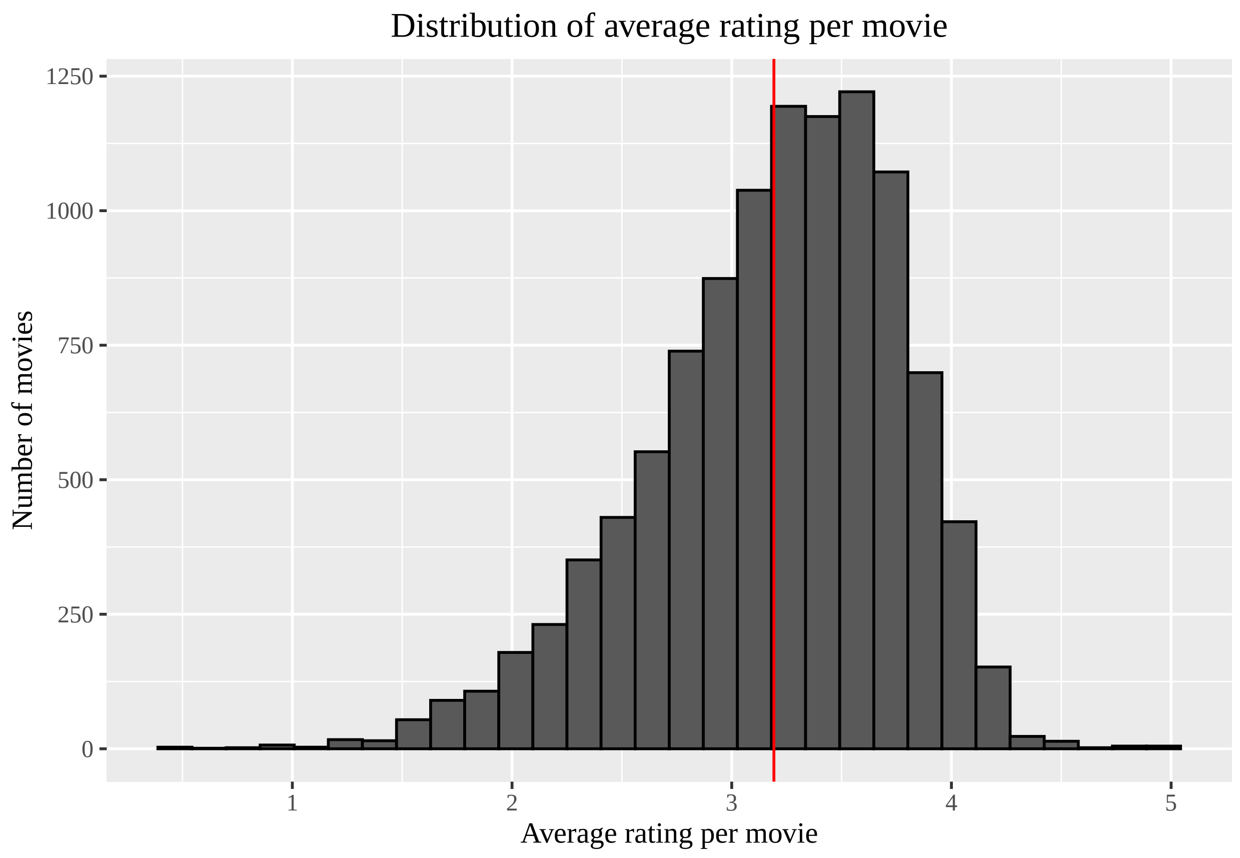
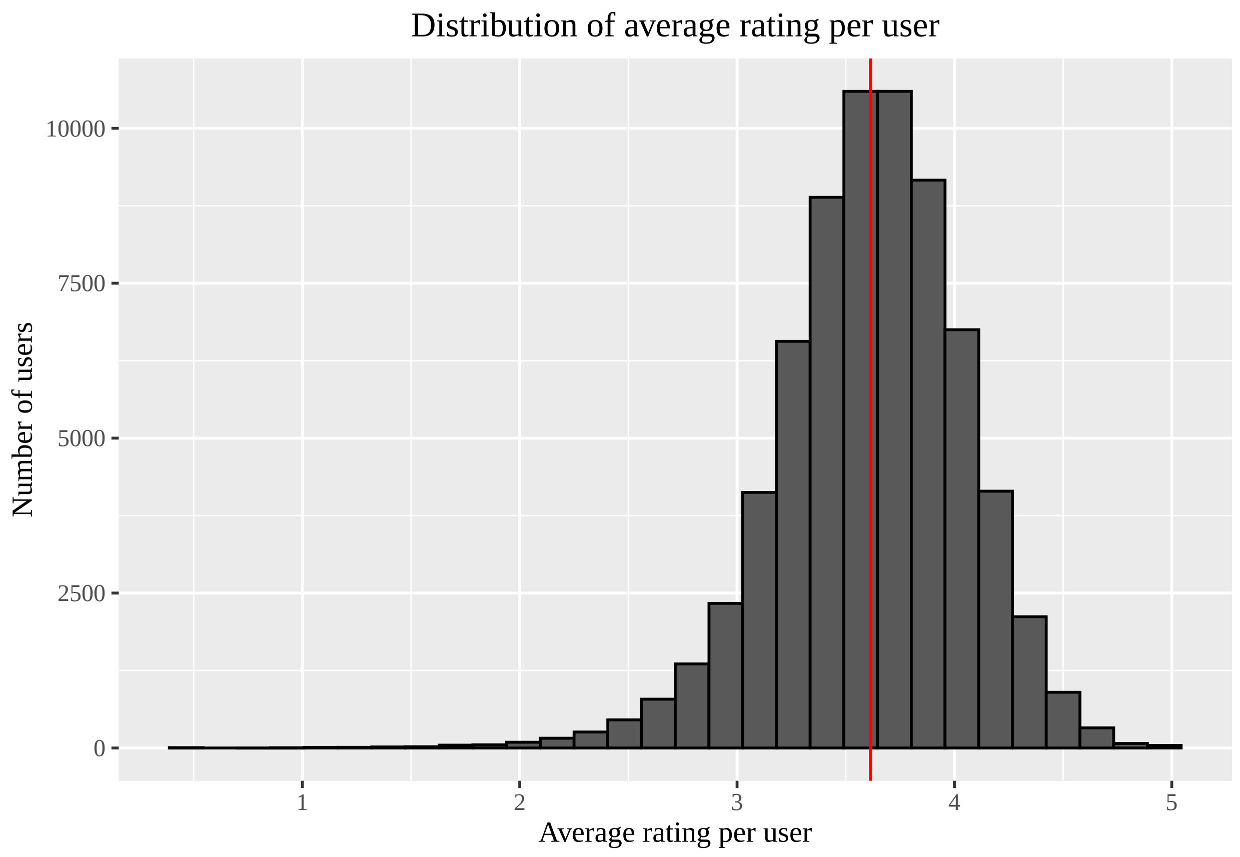
In a first step, I had a look at the total number of unique users, the total number of unique movies and the average rating in the MovieLens dataset.

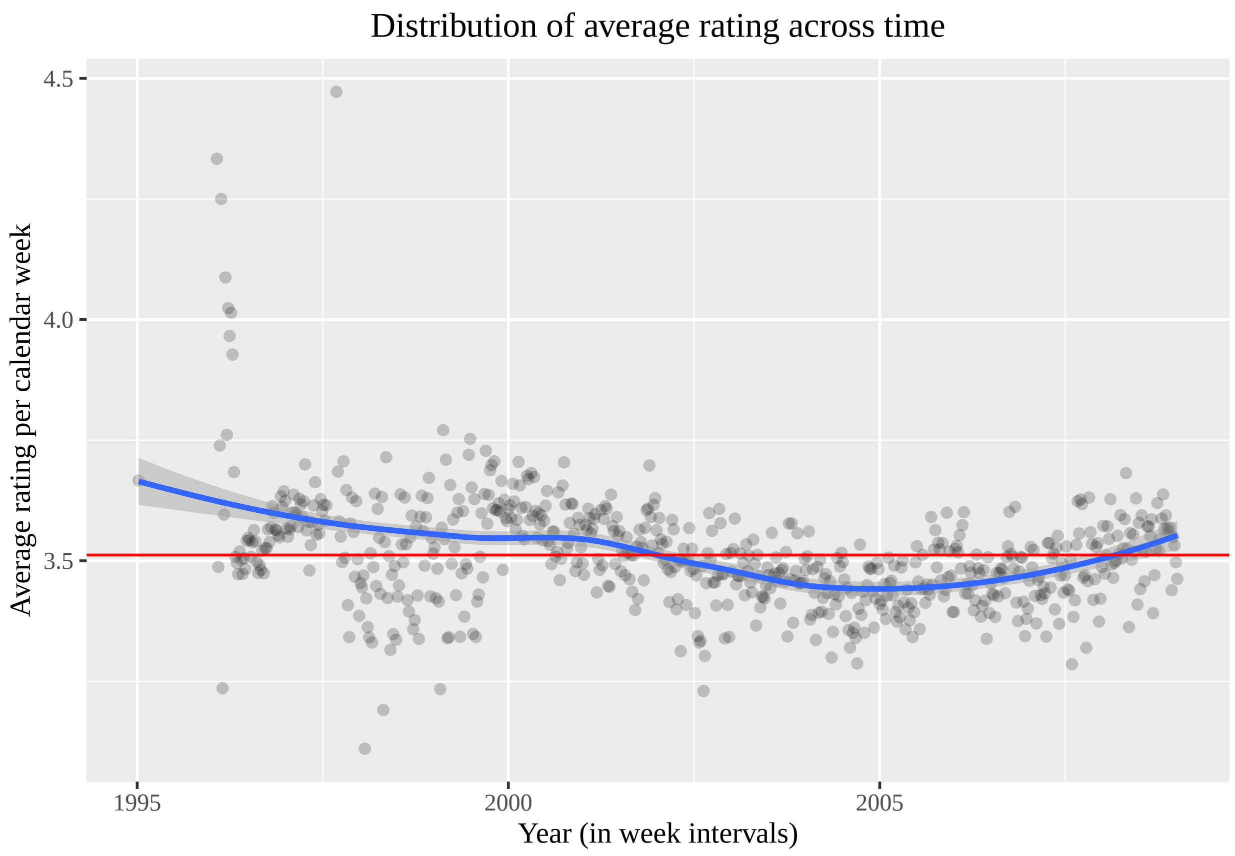
```r
unique_users = length(unique(movielens$movieId))
unique_movies = length(unique(movielens$userId))
average_rating = mean(movielens$rating)
```

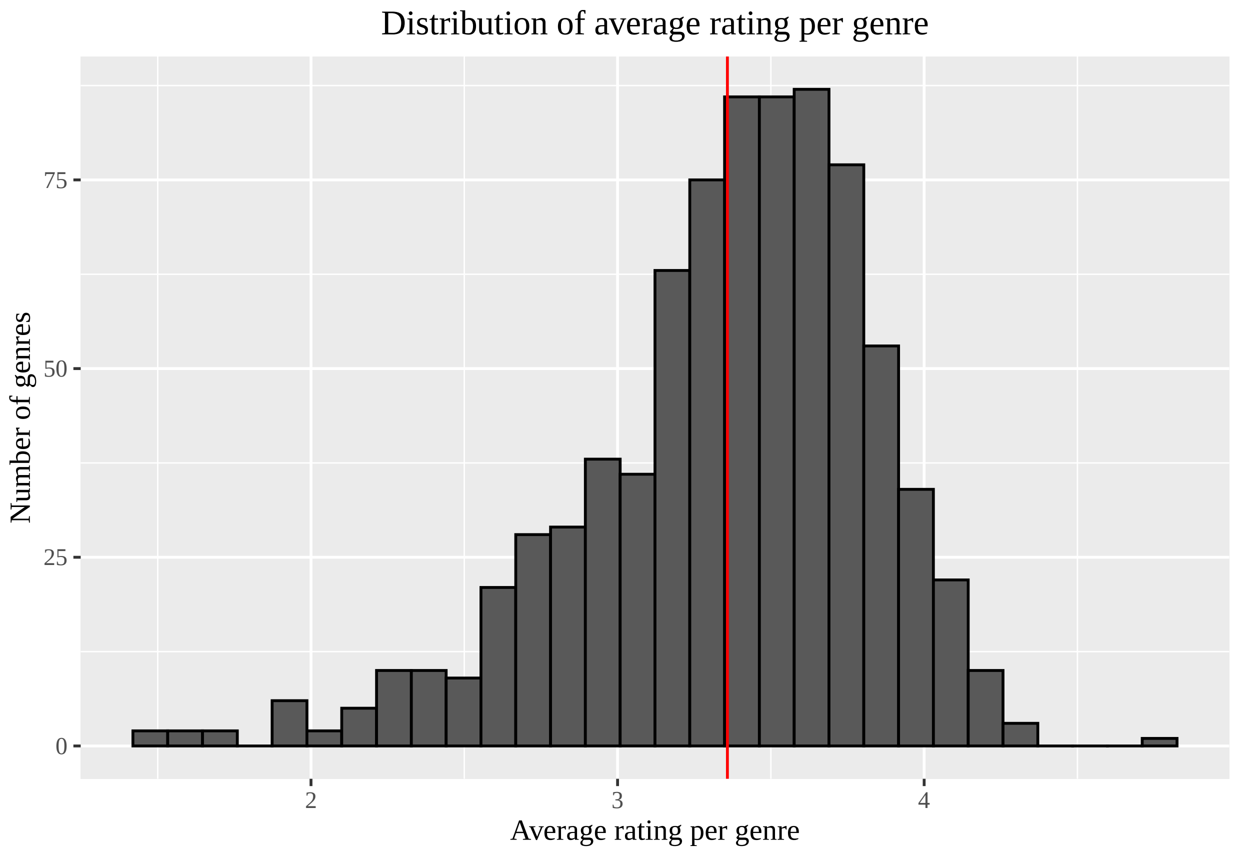
The number of unique users is 10677, the number of unique users is 69878 and the average rating is 3.51242.

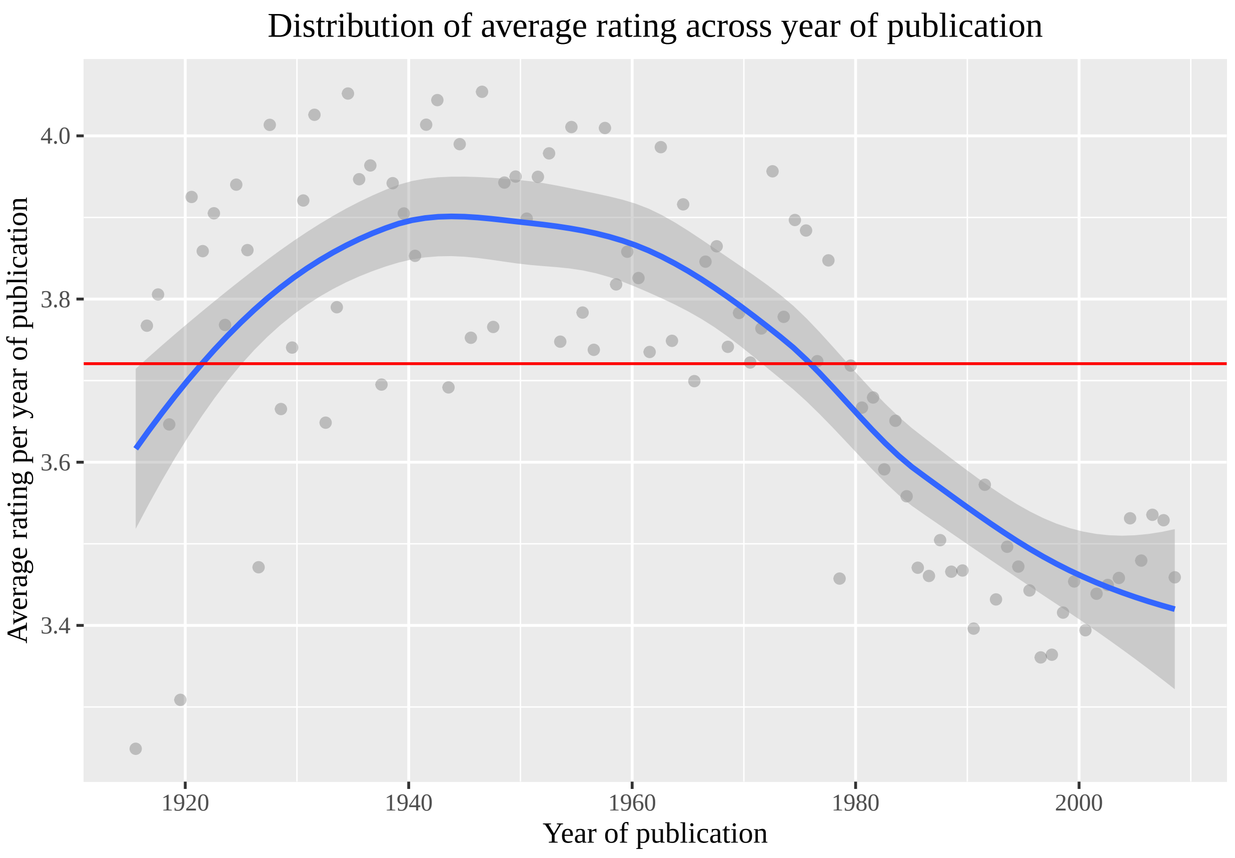
As we learned in class about biases, I wanted to get a better understanding of potential noisy factors that influence movie ratings. By looking at the column names of the MovieLens dataset, I could identify five potential factors that influence movie ratings: the user, the movie, the year the movie was published, the movie genre(s) and the week the movie was rated. Thus, I decided to inspect the rating distribution of those five factors in more detail.

For every factor, I grouped its elements and calculated the average rating for each of the groups. Afterwards, I visualized the results to get a better understanding of the distribution and included the "averaged average" (red line). In the following, the results of this data exploration are visualized.


Distribution of average rating per movie
Number of movies
1250
1000
750
500
250
0
1
2
3
4
5
Average rating per movie

Distribution of average rating per user
Number of users
10000
7500
5000
2500
0
1
2
3
4
5
Average rating per user


Distribution of average rating across time
4.5
4.0
3.5
Average rating per calendar week
1995
2000
2005
Year (in week intervals)


Distribution of average rating per genre
75
50
25
0
Number of genres
2
3
4
Average rating per genre



For all five factors, biases are clearly discernible: some movies are rated better than others, some users are more cranky than others, average ratings tended to decrease until 2005, some genres are more popular than others and old movies tend to better than new ones. Consequently, an appropriate machine learning algorithm needs to take into account all of those factors.

Modeling approach

To model the movie recommendation system, I decided to start with the simplest possible recommendation system and adjusted it factor by factor, based on the insights described in the previous section. For the adjustment, I grouped each factor's elements and calculated each group's average rating. To receive the respective bias value, b_x, I took the average of the distance between the true rating in the train set and the predicted rating by the algorithm *(as demonstrated later in the code for the movie + user + time + genre + publish effect model)*. The formula of the resulting model is

$$Y_{i,u,t,g,p} = \mu + b_i + b_u + b_t + b_g + b_p + \epsilon_{i,u,t,g,p}$$

with $Y_{i,u,t,g,p}$ respresenting user u's rating at time t for movie i of genre g, published in year p.

Due to the large amount of unique users and unique movies, there were some movies that have just been rated by a very low number of users, bearing the risk that the predictions based on this data are significantly distorted. For this reason, I performed regularization in a last step to fine tune the model. Regularization limits the variability of effect sizes by penalizing large estimates from small sample sizes.

In total, there were seven different models incorporating the different aspects of the modeling approach:

1. Average model
2. Movie effect model
3. Movie + user effect model
4. Movie + user + time effect model
5. Movie + user + time + genre effect model
6. Movie + user + time + genre + publish effect model
7. Regularized movie + user + time + genre + publish effect model

Each model was trained using the train set. After training the model, the ratings in the test set were predicted and the resulting RMSE calculated. Comparing the resulting RMSEs, it was apparent that the more effects were included, i.e. the more complex the model got through including noisy factors, the better the RMSE. As soon as the RMSE reached a level sufficient to my expectations, I applied the resulting model to the validation set to receive the final RMSE of my machine learning algorithm.

Average model

The cornerstone of the algorithm is the average model. It is the simplest possible recommendation system and predicts the same rating for all movies, regardless of the movie, user, time of rating, genre or year of publication. The predicted rating is simply the average rating across all movies and all users in the train set.

```r
# average rating across all users
mu <- mean(train_set$rating)

# use mu to compute RMSE on the test set
mu_rmse <- RMSE(mu, test_set$rating)

# use mu_rmse to create rmse_results table
rmse_results <- data_frame(method = "Average Model", RMSE = mu_rmse)
```

The average model yields a RMSE of 1.06114 and leaves significant room for improvement.

Movie + user + time + genre + publish effect model

To improve the average model, I included the five different factors one after another. In this report, I only show the code for incorporating the last factor, the publish effect, into the model.

```r
# calculate publish effect
publish_avgs <- train_set %>%
  left_join(movie_avgs, by ='movieId') %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(week_avgs, by = "date") %>%
  left_join(genre_avgs, by = "genres") %>%
  group_by(year) %>%
  summarize(b_p = mean(rating - mu - b_i - b_u - b_t - b_g))

# calculate predicted ratings when additionally incorporating the genre effect
publish_effect_predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  left_join(week_avgs, by = "date") %>%
  left_join(genre_avgs, by = "genres") %>%
  left_join(publish_avgs, by = "year") %>%
  mutate(pred = mu + b_i + b_u + b_t + b_g + b_p) %>%
  .$pred
```

```r
# use predicted ratings by incorporating publish effect to compute RMSE on the test set
publish_effect_rmse <- RMSE(test_set$rating, publish_effect_predicted_ratings)
```

After incorporating all effects into the model, the RMSE results table reveals that the algorithm was significantly improved compared to the initial average model.

```r
kable(rmse_results)
```

method	RMSE
Average Model	1.06114
Movie Effect Model	0.94416
Movie + User Effect Model	0.86597
Movie + User + Time Effect Model	0.86589
Movie + User + Time + Genre Effect Model	0.86552
Movie + User + Time + Genre + Publish Effect Model	0.86531

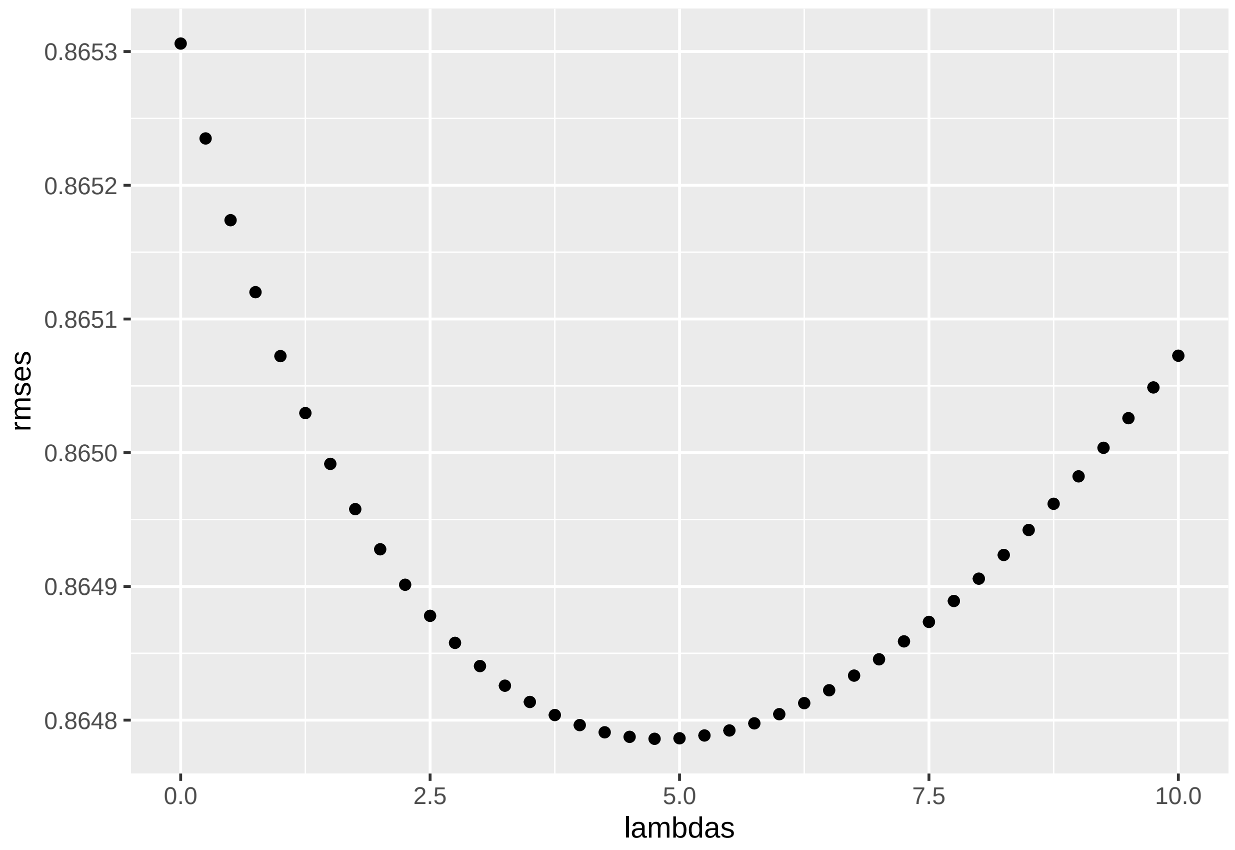
Regularized movie + user + time + genre + publish effect model

In a last step, I performed regularization as described in the modeling approach. Instead of minimizing the RMSE (error term ϵ), I minimized the following equation (penalty term λ):

$$\frac{1}{N}\sum_{u,i,t,g,p}(Y_{i,u,t,g,p}-\mu-b_i-b_u-b_t-b_g-b_p)^2+\lambda(\sum_i b_i^2+\sum_u b_u^2+\sum_t b_t^2+\sum_g b_g^2+\sum_p b_p^2)$$

The optimal lambda that minimizes the penalty term and the equation can be identified graphically. The optimal value of lambda is 4.75.

```r
qplot(lambdas, rmses)
```



After performing regularization, I was able to further reduce the RMSE on the test set to 0.86479. I considered this RMSE sufficient for a good and robust movie recommendation system and thus moved on to apply my movie recommendation system to the validation set.

Results

```
# calculate different regularized effects on whole edx set (before only test set)
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
user_reg_avgs <- edx %>%
  left_join(movie_reg_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda), n_u = n())
time_reg_avgs <- edx %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  left_join(user_reg_avgs, by = 'userId') %>%
  group_by(date) %>%
  summarize(b_t = sum(rating - b_u - b_i - mu)/(n()+lambda), n_t = n())
genre_reg_avgs <- edx %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  left_join(user_reg_avgs, by = 'userId') %>%
  left_join(time_reg_avgs, by = 'date') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_t - b_u - b_i - mu)/(n()+lambda), n_g = n())
publish_reg_avgs <- edx %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  left_join(user_reg_avgs, by = 'userId') %>%
  left_join(time_reg_avgs, by = 'date') %>%
  left_join(genre_reg_avgs, by = 'genres') %>%
  group_by(year) %>%
  summarize(b_p = sum(rating - b_g - b_t - b_u - b_i - mu)/(n()+lambda), n_p = n())

# use the regularized model to calculate predicted ratings on the validation set
validation_predicted_ratings <- validation %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by = "userId") %>%
  left_join(time_reg_avgs, by = "date") %>%
  left_join(genre_reg_avgs, by = "genres") %>%
  left_join(publish_reg_avgs, by = "year") %>%
  mutate(pred = mu + b_i + b_u + b_t + b_g + b_p) %>%
  .$pred

# calculate final RMSE on validation set
validation_rmse <- RMSE(validation$rating, validation_predicted_ratings)
```

When applying my final algrorithm, the regularized model, on the validation set, I was able to achive a **RMSE** of **0.86413**. This is the final RMSE of my movie recommendation system based on my own machine learning algorithm.

Conclusion

method	RMSE
Average Model	1.06114
Movie Effect Model	0.94416
Movie + User Effect Model	0.86597
Movie + User + Time Effect Model	0.86589
Movie + User + Time + Genre Effect Model	0.86552
Movie + User + Time + Genre + Publish Effect Model	0.86531
Regularized Movie + User + Time + Genre + Year Effect Model	0.86479
Final RMSE on Validation Set	0.86413

By incorporating five different bias into my algorithm and by performing regularization, I was able to significantly reduce the RMSE of the initial average model by 1.06114 to 0.86479 (RMSE of the regularized model on the test set). Applying the algorithm onto the validation set yielded a final RMSE of 0.86413.

In the future, I aim to further improve my movie recommendation system by modeling the time and publish effects with smooth functions. Furthermore, the genres of each movie could be treated in an improved fashion by grouping movies into similiar clusters (that share some but not all genres), not only identical clusters.