edX Movielens Project

Felix Kleinpeter

4 2 2022

Contents

1 Executive Summary

This is a report about building a movie rating prediction model based on data given in the MovieLens dataset. This is a submission for the capstone project of HardvardX's data science course. During the project, I first extracted the data, explored and visualized it, and then assumed reasonable influencing factors for my rating prediction model based on what was found in the exploration and visualization. After gaining these insights into the data, I split the edx subset of the data into a train and test set which I used to create different iterations of a rating prediction model. This model took into account factors like the individual movies and users, the genres, release year as well as time difference between release and rating. Using a regularized version of a model containing these factors, I was able to reduce RMSE to 0.8639332 in a final hold out test against the validation set, which thus achieved the target of an RMSE value of 0.86490 or less.

2 Introduction and Setup

This submission for the final project of the HarvardX Data Science Course is going to analyze the MovieLens dataset as well as create a movie recommendation system based on these data. The final goal is to train a machine learning algorithm with the inputs from one subset of the data to predict movie ratings in another (validation) subset of the data and minimize the prediction loss denoted in Root Mean Squared Error.

To set up the analysis, I first download and divide up the data using the code given by the HarvardX course here.

The provided code splits the MovieLens data set into the edx subset, which is used for developing the machine learning algorithm, and the validation subset, which is used for the final test of the algorithm. Root Mean Squared Error (RMSE) is going to be the indicator used to evaluate the accuracy of the algorithm. As given, the data is split into 6 columns, these are:

Table 1: Columns in the MovieLens data

column	description
userId	Unique identifier of the user giving the rating
movieId	Unique identifier of the movie
rating	Rating given by the user on a scale from 0.5 to 5
timestamp	Timestamp of the rating, given in miliseconds since January 1, 1970
title	Title of the movie (release year in brackets)
genres	Combination of genres the movie is categorized into

To make the timestamp more readable, I then use the lubridate package to convert the information in the timestamp column into a more easily readable format in a new date column. More colums will be added in the course of the report as they are needed to extract new information.

3 Data Exploration & Visualization

In this section, I am going to lay out some basic features of the created edx data set which may later influence some ideas that may be used to create the prediction model.

3.1 Basic exploration

There are 9000055 rows in the edx data set which are split into ratings of 10677 distinct movies by 69878 individual users. While this seems like a vast amount of data at the first glance, the given data is actually very sparse. If every user had rated every single movie in the data set, there should be 746087406 rows - the given entries only make up 1.21% of that full matrix!

3.2 Individual Movies

Given that individual movies are of differing qualities, I would expect to see a difference in the average rating per movie. Some movies are very good and also very well received while probably the majority of movies is rather mediocre or even really bad. The average ratings should show this relationship. Additionally, some movies are very popular and well known while others are niche movies that maybe only a handful of fans would know. Thus, I would expect a similar situation where a few movies receive the majority of ratings while most movies will have only very few ratings.

Fig. 1: Average Movie Ratings

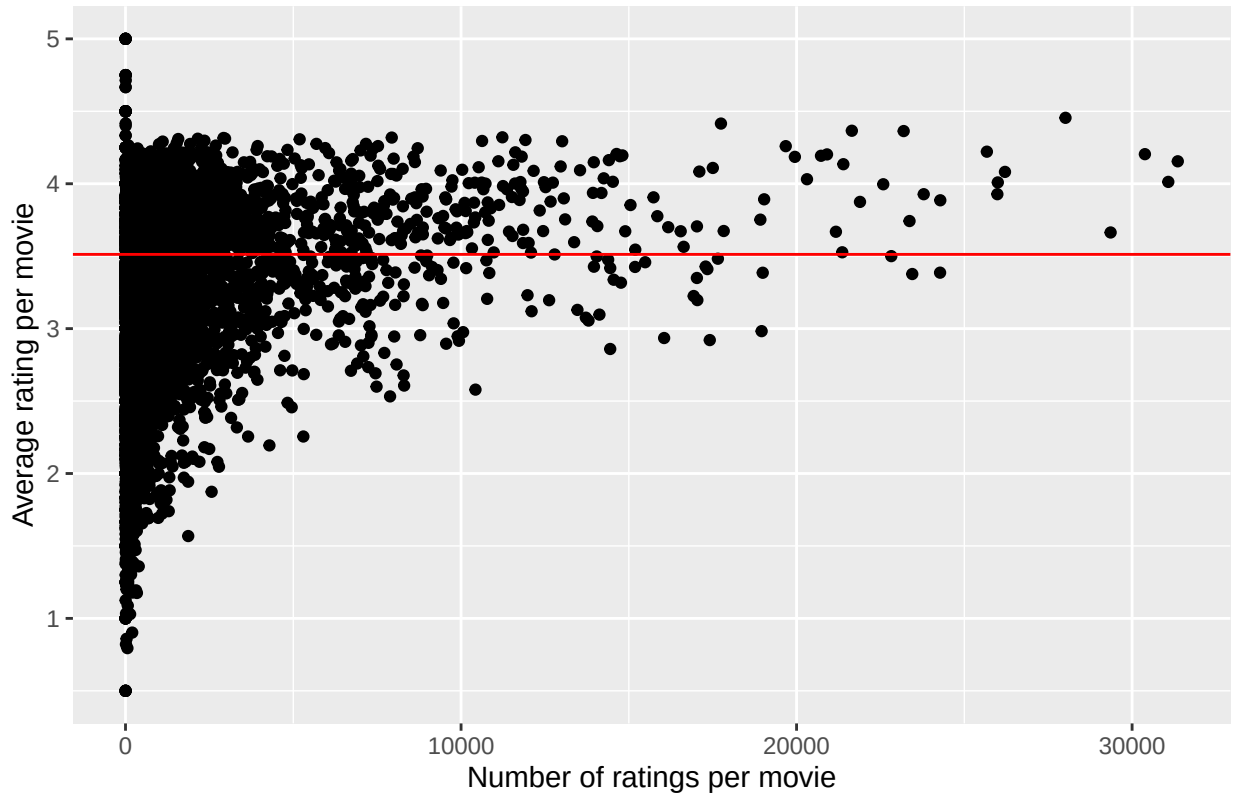


Fig. 1 confirms my expectations about the relationship of individual movies and ratings: * the vast majority of movies has a relatively low number of ratings while a few movies have tens of thousands of ratings * almost

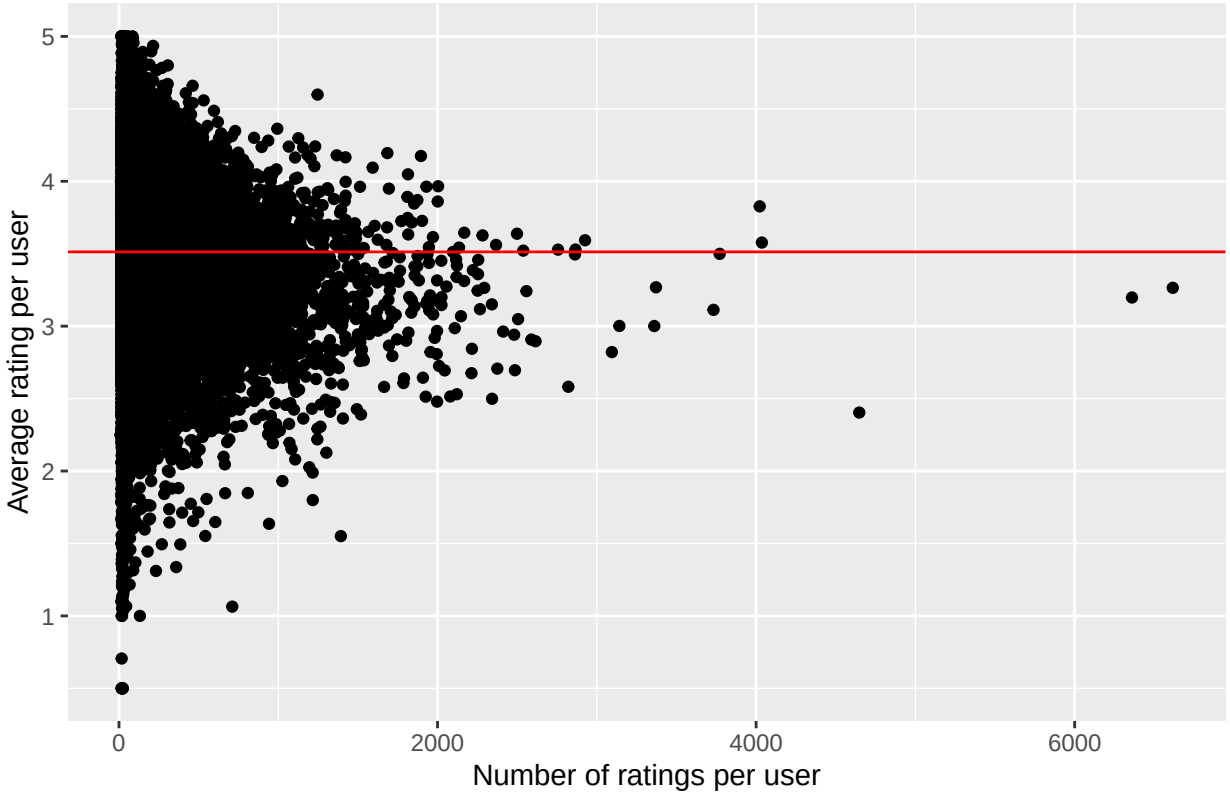
all of the movies with more than 20'000 ratings have an average rating that is higher than the average rating across all movies (red line in the graph) * the movies with the highest and lowest average ratings have very few ratings, in some cases maybe even as low as a single rating

Thus, the individual movie certainly seems like a valid factor for the prediction model that will be built in chapter 3.

3.3 Individual Users

Similar to the variety of movies, there are also a variety of people watching these movies: some people are "movie buffs" who watch and rate a very high number of movies in their free time (or might even be professional movie critics), while other people only watch movies here and there and thus would be expected to rate a lower number of movies. Equally, some people are generally more positive in their evaluations while others are more strict. I expect the following graph to show a wide range of average ratings per user, reflecting the described behavioral differences.

Fig. 2: Average User Ratings



Again, the graph in figure 2 confirms the existing suspicions: * average ratings per user vary widely * the majority of the users has relatively few ratings while a few users have rated multiple thousands of movies

3.4 Movie genres

The movies in the data set are distributed in different genres. Movies can also be classified into a mix of multiple genres. I will first explore the number of genres as a possible predictive factor, then look at the

actual given genre combinations. To better understand how granular the genres are, I create a list of the genres that make up the combinations included in the edx sample:

Table 2: Genres in the MovieLens data

genres
Comedy
Drama
Thriller
Western
Horror
Documentary
Action
Romance
Sci-Fi
Children
Adventure
Animation
Musical
Film-Noir
Crime
War
Mystery
Fantasy
IMAX
(no genres listed)

For the combinations of genres, I intuitively assume that most movies would either fit into a single genre or be a crossover of a maximum of three different genres as crossovers with more themes are rarely produced. Additionally, I assume that the average rating of movies with up to 3 genres is higher than the average rating of movies that are a crossover of more than 4 genres. This assumption is based on the idea that movies that fall into too many genres might be perceived as confusing or too ambitious by the audience, thus receiving worse ratings.

Fig. 3: Genres per movie

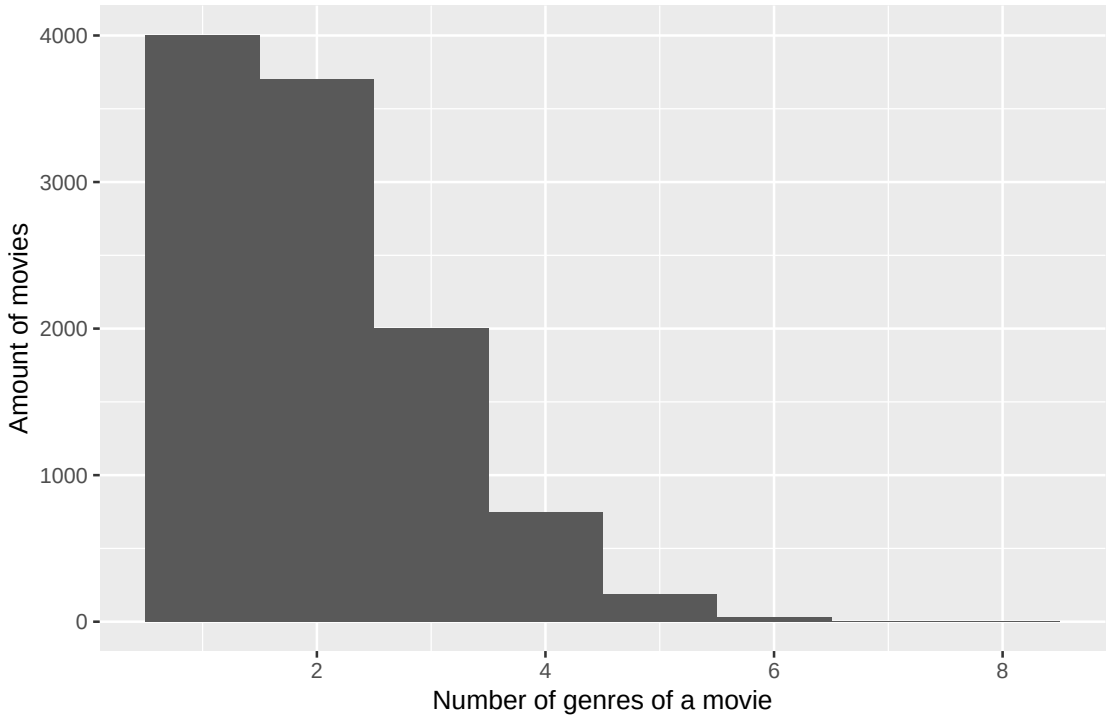


Fig. 4: Average ratings per numbers of genres

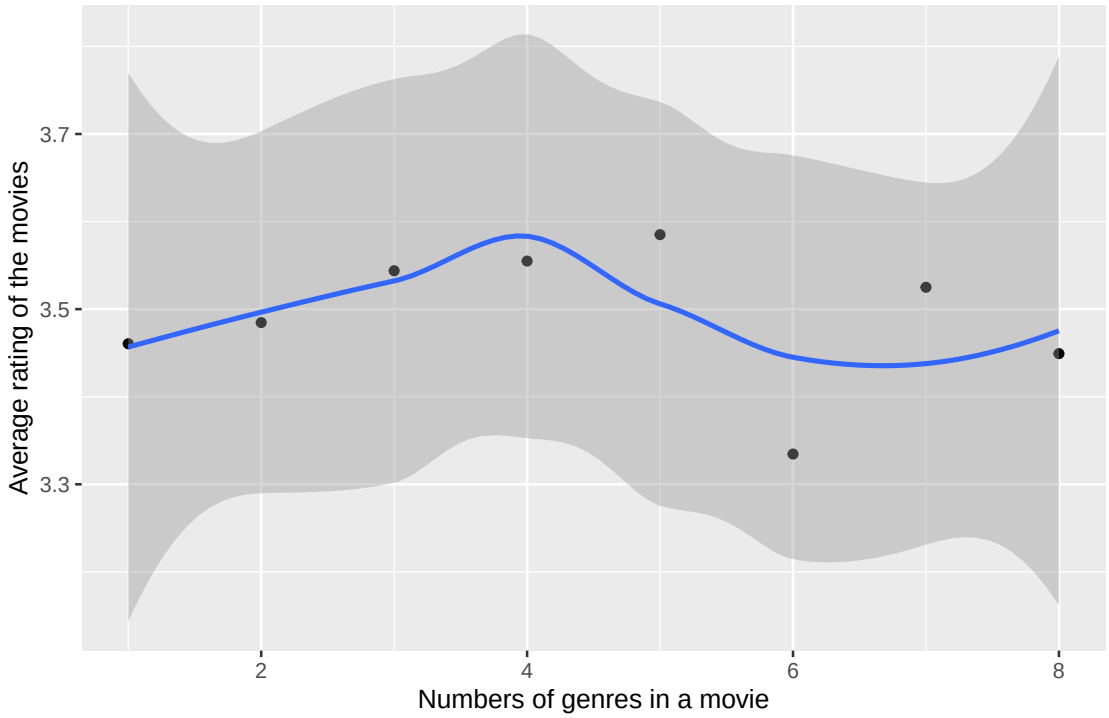


Figure 3 clearly shows that - as expected - most movies are categorized distinctly into one or few genres. The second expectation stated above is rejected by Figure 4 though: average ratings keep rising until a crossover of five genres is reached. It has to be said though that with the exception of one outlier at six genres (which,

as Figure 3 shows, is based on a small number of movies), the averages still all fall between roughly 3.45 and 3.6, meaning that while there is an effect of the number of genres on the average rating, the effect seems to be rather small.

Secondly, now I will look at the average rating of movie genres. To do this, I will return to the given combinations of individual genres, instead of the 20 base genres.

As there are 797 combinations of genres in the sample, I will focus on the top 20 most prevalent combinations first to see if different genres have different average ratings.

Fig. 5: Average rating per genre combination

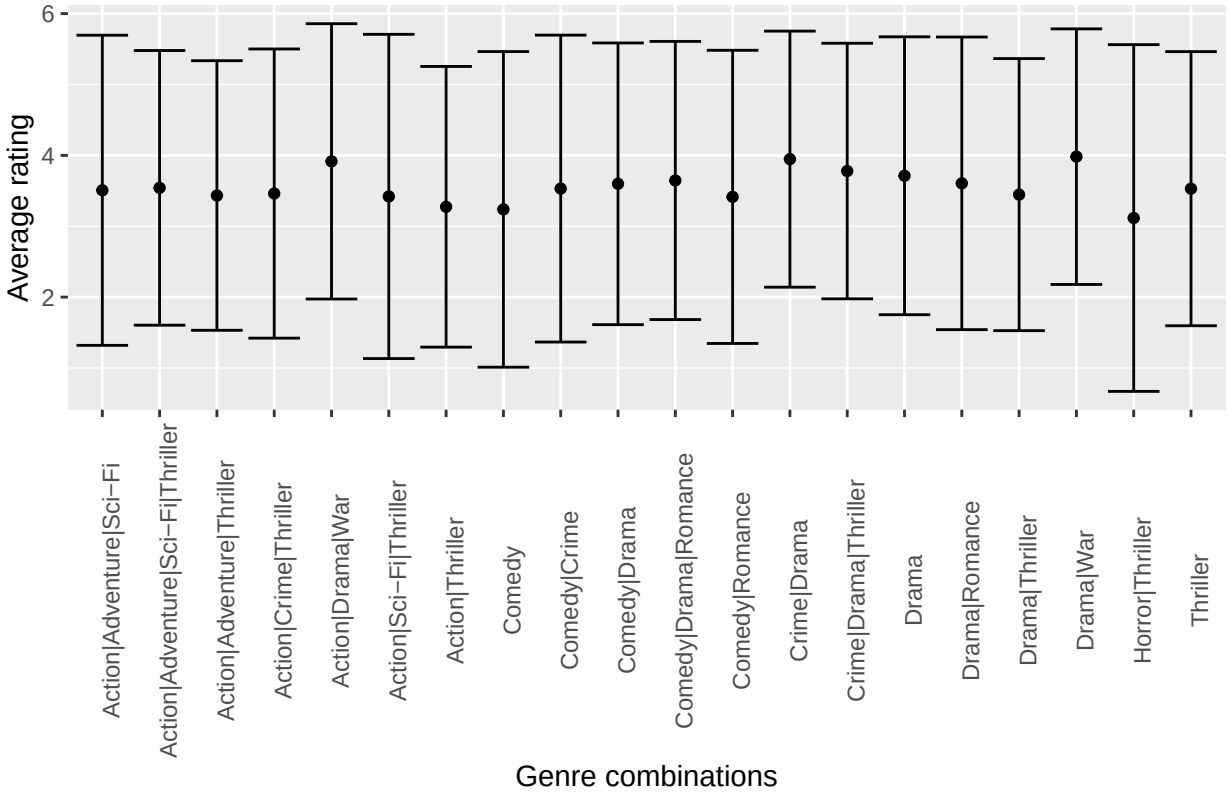


Figure 5 shows a clear indication that even among the most rated genre combinations there are quite different average ratings per genre - from as little as just over 3 for the Horror|Thriller category to around 4 for Drama|War and Action|Drama|War. Additionally, the different lengths of the error bars indicates that ratings within some genres (for example the Drama|War category) are more uniform than in others (for example the Horror|Thriller category).

Both angles of looking at the given genre data invite the idea to later include a genre factor into the prediction model. In order to keep the model simpler, I will pick only one of the two angles though. As the differences between genre combinations are generally larger than the differences between the movies of a different number of genres, I will pick the exact genre combinations as a factor for the model.

3.5 Movie Release Year

Another piece of information that is given in the data is the movie release year. To use it in this analysis, I first have to extract it from the "title" column and turn it into a numerical value. Based on this data, I can attempt to answer an interesting question given the increasing amount of sequels and remakes in the recent years though: was there an increase or decrease in the (perceived) quality of movies made over time?

Fig. 6: Average movie rating by release year

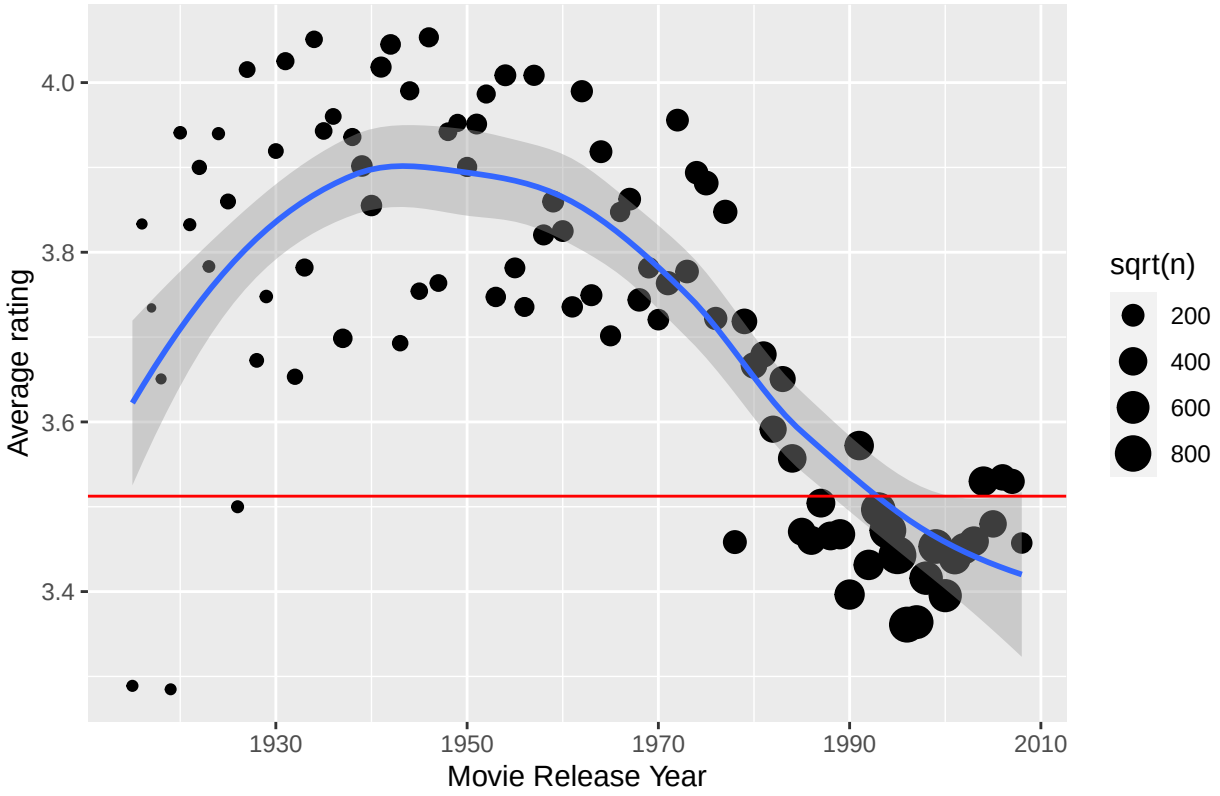


Figure 6 shows an impressive trend: for almost every year before 1980, the average ratings for movies released in this year is higher than the overall average rating (with a smoothed out peak in the 1940s), whereas movies made since the 1980s are mostly below average. Interestingly, the distance to the overall average (denoted by the red line) is much smaller for the newer movies as the sheer amount of ratings (denoted by the size of the points) has a much stronger effect on the overall arithmetic mean rating than the few ratings that were received by older movies. Nevertheless, the differences in average rating between release years are so significant, they will become a factor in the final prediction model.

3.6 Movie ratings over time

Another data point that is given with each of the ratings in the edx data set is the timestamp of the rating. In this section, I'm going to investigate if there is a time effect on the ratings (i.e. are people getting stricter or nicer with their judgments) to see if a time factor should be used with the final model. Potentially, there could also be an in-year factor (for example more favorable ratings around Christmas).

Fig. 7: Average ratings by review year

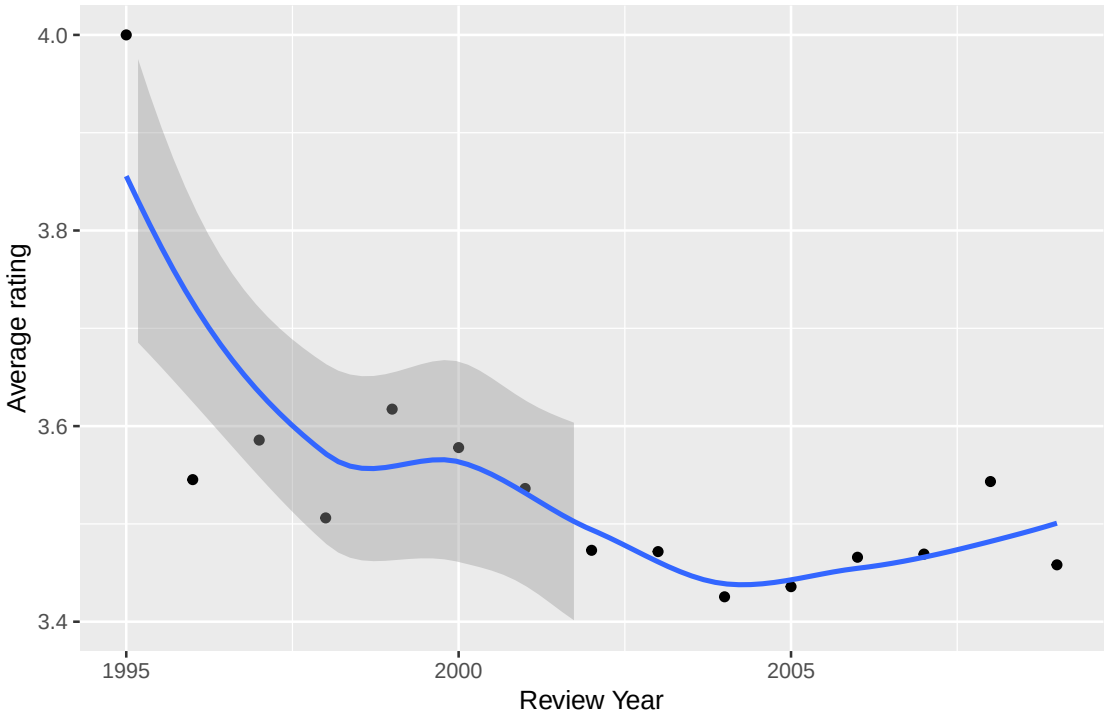
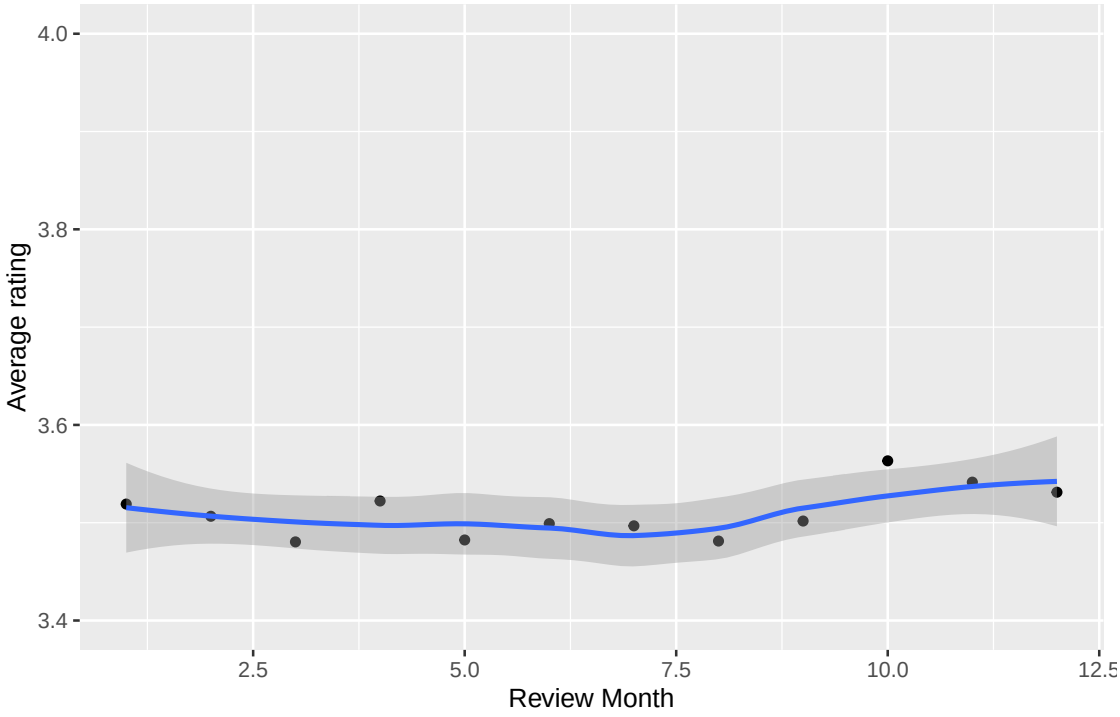


Fig. 8: Average ratings by review month



Both Figures 7 and 8 show that there is some effect of the review time on the average ratings, with the one found in Figure 7 when looking at the years being a bit stronger than the in-year effect.

Additionally, I can see if there is an effect from the distance between the time of review and the release of the movie. The intuitive explanation here would be that if someone rates a movie right after release, they might be biased by hype or marketing at the time. This should cancel out over the next years and possibly yield lower movie ratings when the review happens 2+ years after the release of the movie. And finally, there might be an increase in average rating again for very long timeframes between release and review. This is because when someone now watches (and reviews) a very old movie, this is usually not because of a random choice of movie, but because this movie is a "classic" or "has stood the test of time", so would be expected to be an above-average movie.

In setting up the data, there are 175 cases where the time difference between review year and release year of the movie is negative. These might be the result of reviewers getting early access before an actual release date, differing release dates across the globe or simply some erroneous data points. As these only make up 0.00194% of the data, the impact of these logical errors should be negligible. To remove the logical error, I define any "negative" time difference to be zero instead, i.e. rating in the same year as movie release. This then leads to the following plot for the relationship between time difference and average ratings:

Fig. 9: Average rating based on time between release and review

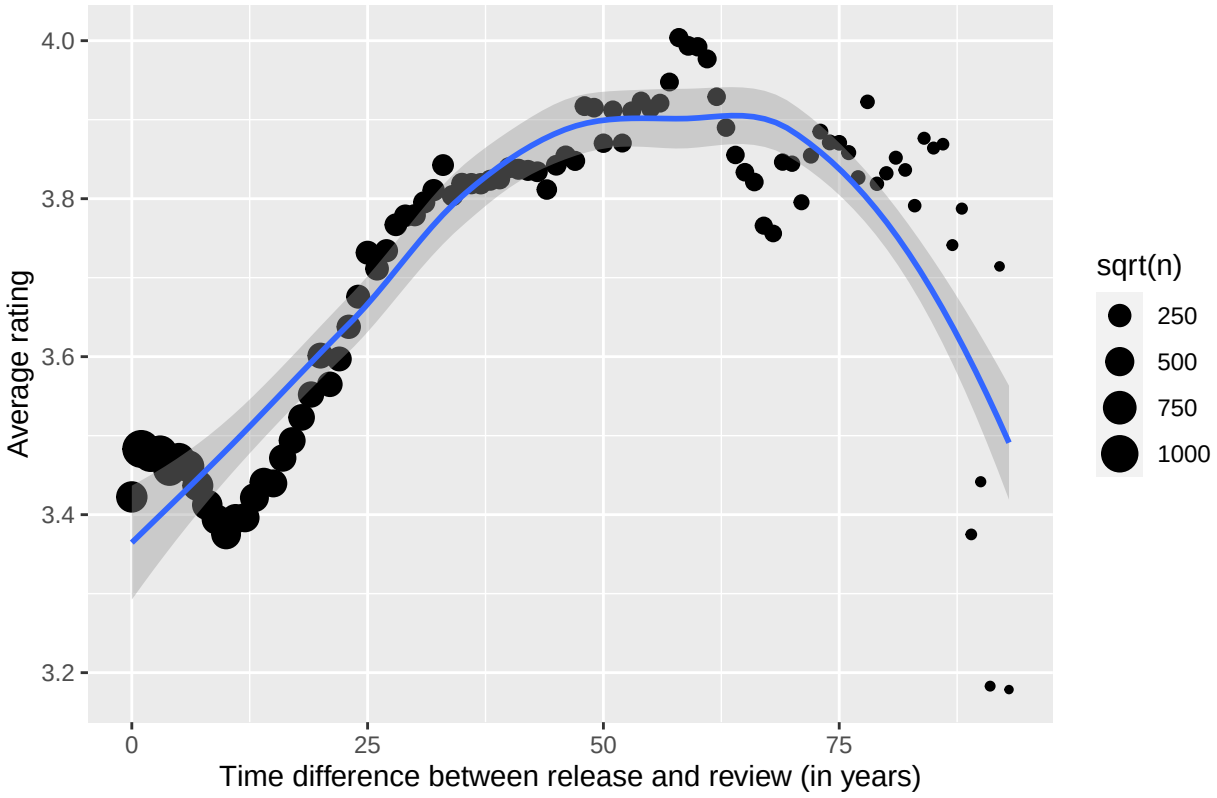


Figure 9 confirms our expectations about the behavior of ratings related to the time difference between rating and movie release. There is an observable drop in average rating once the time difference rises above roughly 90 years, however the the size of the points in the graphs shows that there is a comparatively small sample size of ratings for these time differences, thus this drop might be caused by only a few outliers. As this measure shows a stronger effect than the metrics observed in Figures 7 and 8, I will use this time difference between release and review as the fifth factor in my prediction model.

4 Modeling

In this chapter, I am going to explain step-by-step the building of the model that will lead to the final movie recommendation model. To do so, I create a test set and a train set to evaluate the accuracy of the different models and to choose a final model which will be put to the test against the validation set in chapter 4. Contrary to chapter 2, which only showed the code outputs, I will show the actual code chunks used to create the models here in chapter 3.

```
# Set seed to have consistent results
set.seed(1910, sample.kind = "Rounding") # if using R 3.5 or earlier, use set.seed(1910)
# Create train (80%) and test (20%) sets
test_set_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_set_index,]
test_set <- edx[test_set_index,]

# clean up test set to not include users or movies that are not in the train set
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# define target RMSE
target_rmse <- 0.86490
```

4.1 Basic model

For the basic model, I start with the simplest possible assumption: every rating consists of the average rating μ and an error term $\varepsilon_{u,i}$ centered around 0.

```
# Calculate average rating in train set
mu_hat <- mean(train_set$rating)
# Calculate RMSE in test set
basic_rmse <- RMSE(test_set$rating, mu_hat)
# add result to table
rmse_results <- tibble(model = c("Target RMSE", "Basic average"),
                       RMSE = c(target_rmse, basic_rmse),
                       difference = c(0, basic_rmse - target_rmse))
```

With this very naive model, I am able to reach a RMSE value of 1.0603677. This is of course still a long way from the target of 0.86490. However, it is a valid starting point to add other factors into the model.

4.2 Adding a movie factor

As the visualization in chapter 2 showed, different movies receive very different average ratings, so this should be reflected in the prediction model. The movie factor is going to be denoted as b_i and added to the basic model. This creates the following updated model:

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

with the average μ, the movie specific factor b_i and the error term $\varepsilon_{u,i}$. As we know that in this case the least squares estimate for b_i is equal to the average of $Y_{u,i} - \mu$, the estimates for b_i can be found through the following code:

```
# create movie specific factor
movie_avgs <- train_set %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu_hat))
```

Having created the movie specific factors, I can now add them to the basic simple average model to create a new set of predictions and evaluate the new model by calculating the resulting RMSE.

```
# add b_i to predictions
predicted_ratings <- mu_hat + test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  pull(b_i)
# calculate RMSE
b_i_rmse <- RMSE(predicted_ratings, test_set$rating)
# add new model RMSE to summary table
rmse_results <- bind_rows(rmse_results,
                          tibble(model = "Movie Effect", RMSE = b_i_rmse,
                                 difference = b_i_rmse - target_rmse))
```

Adding the movie-specific factor thus already drops the RMSE down to 0.944377 from the previous 1.0603677

4.3 Adding a user factor

As seen in the exploratory data analysis in chapter 2, there is a good reason to believe that there should be a user factor in the model as average ratings per user vary quite strongly. This would now yield the model

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

with the average μ, the movie specific factor b_i the user specific factor b_u and the error term $\varepsilon_{u,i}$. Again, the least squares estimate for b_u would be equal to the average of $Y_{u,i} - \mu - b_i$. Hence, I'll use the following code to add the user factor to the existing model:

```
# create user specific factor
user_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))
```

Having created these approximations for the user factor b_u, I can now generate new predictions for the two-factor model.

```
# add b_u to predictions
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu_hat + b_u + b_i) %>%
  pull(pred)
# calculate RMSE
add_b_u_rmse <- RMSE(predicted_ratings, test_set$rating)
# add new model RMSE to results table
rmse_results <- bind_rows(rmse_results,
                          tibble(model = "Movie + User Effect", RMSE = add_b_u_rmse,
                                 difference = add_b_u_rmse - target_rmse))
```

The addition of the user factor b_u thus results in another improvement to the RMSE to now 0.8664927. However, although this is significantly better than the basic model's RMSE, it is still quite far away from the target of 0.8649.

4.4 Genre Factor

The third factor I identified in chapter 2 as a potentially useful factor for the prediction of ratings is the genre, which will be denoted as b_g. Adding this to the existing model with movie and user factors, the new model is defined as:

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

with the average μ, the movie specific factor b_i the user specific factor b_u, the genre specific factor b_g and the error term $\varepsilon_{u,i}$. Similar to the previous factors, I estimate the genre specific factor as the average distance between the actual ratings in the test set and the result given the existing factors, i.e. the mean of $Yu, i - \mu - b_i - b_u$.

```
# create genre specific factor
genre_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u))
```

Having created the factor, it is again time to then make predictions based on this new model and calculate the corresponding RMSE.

```
# add b_g to predictions
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  summarize(pred = mu_hat + b_i + b_u + b_g) %>%
  pull(pred)
# calculate RMSE
add_b_g_rmse <- RMSE(predicted_ratings, test_set$rating)
# add RMSE to table
rmse_results <- bind_rows(rmse_results,
                          tibble(model = "Movie + User + Genre Effect",
                                 RMSE = add_b_g_rmse,
                                 difference = add_b_g_rmse - target_rmse))
```

The resulting RMSE from the three-factor model containing movie, user and genre factors end up at 0.8661672, which is a very small improvement over the previous result.

4.5 Release Year Factor

Adding a fourth factor into the mix, I will now have a look at the release year of the rated movies. The exploratory analysis in chapter 2 showed a clear downward trend for ratings in relationship to the release year, so this might yield a stronger model improvement than the genre factor before. I will stick with my way of estimating the effects of the new factor b_{rel}, defining it as the mean difference per year between the actual rating and the predicted rating as given by the previous factors, i.e. the mean of $Yu, i - \mu - b_i - b_u - b_g$. This gives the new model:

$$Y = \mu + b_i + b_u + b_g + b_{rel} + \varepsilon_{u,i}$$

with the average μ, the movie specific factor b_i the user specific factor b_u, the genre specific factor b_g, the release year factor b_rel and the error term $\varepsilon_{u,i}$. Again, I first calculate the factor values for each of the release years:

```
# create release year specific factor
release_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  group_by(RelYear) %>%
  summarize(b_rel = mean(rating - mu_hat - b_i - b_u - b_g))
```

And then I calculate a new set of predictions for the now four-factor model including the new release year information.

```
# calculate four factor prediction values
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  left_join(release_avgs, by = "RelYear") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g + b_rel) %>%
  pull(pred)
# calculate four factor RMSE
add_b_rel_rmse <- RMSE(predicted_ratings, test_set$rating)
# add new model RMSE to table
rmse_results <- bind_rows(rmse_results,
                          tibble(model = "Movie + User + Genre + Release Year Effect",
                                 RMSE = add_b_rel_rmse,
                                 difference = add_b_rel_rmse - target_rmse))
```

Again, the improvements in RMSE are marginal, moving from 0.8661672 of the three-factor model to 0.8660119 of the four factor model.

4.6 Review Time Factor

The last factor that will be added is the review time factor. In the data exploration section I found that the time between release and review seems to be a good indicator, so I will use this for my model. The model thus is now defined as:

$$Y = \mu + b_i + b_u + b_g + b_{rel} + b_t + \varepsilon_{u,i}$$

with the average μ, the movie specific factor b_i the user specific factor b_u, the genre specific factor b_g, the release year factor b_rel, the review date factor b_t and the error term $\varepsilon_{u,i}$.

```
# create time difference specific factor
timediff_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  left_join(release_avgs, by = "RelYear") %>%
  group_by(timediff) %>%
  summarize(b_t = mean(rating - mu_hat - b_i - b_u - b_g - b_rel))
```

One last time, the following code calculates the RMSE given by the new five-factor predictions.

```
# calculate four factor prediction values
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  left_join(release_avgs, by = "RelYear") %>%
  left_join(timediff_avgs, by = "timediff") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g + b_rel + b_t) %>%
  pull(pred)
# calculate four factor RMSE
add_b_t_rmse <- RMSE(predicted_ratings, test_set$rating)
# add new model RMSE to table
rmse_results <- bind_rows(rmse_results,
                          tibble(model = "Movie + User + Genre + Release Year +
                                 Time Difference Effect",
                                 RMSE = add_b_t_rmse,
                                 difference = add_b_t_rmse - target_rmse))
```

The RMSE value of 0.8656829 is still not very satisfying and quite a distance from the target.

4.7 Regularization

Looking at the progression of RMSE across the iterations of the model described in this chapter so far, it seems clear that the marginal gains of adding new factors are declining more and more. Table 3 confirms this by listing the results so far:

Table 3: Factor Models RMSE

model	RMSE	difference
Target RMSE	0.8649000	0.0000000
Basic average	1.0603677	0.1954677
Movie Effect	0.9443770	0.0794770
Movie + User Effect	0.8664927	0.0015927
Movie + User + Genre Effect	0.8661672	0.0012672
Movie + User + Genre + Release Year Effect	0.8660119	0.0011119
Movie + User + Genre + Release Year + Time Difference Effect	0.8656829	0.0007829

A possible explanation for the still fairly big errors in the existing model is that in those models movies with a relatively low number of ratings have a stronger-than-desired impact on the model. This is to say that for example if we have a movie with only one (extreme) rating in the test set, it creates a disproportionately big error when comparing it to the test set. An approach to combat that is to use regularization. Regularization helps penalize extreme predictions that are formed on the basis of small sample sizes of observations. Mathematically, this is done through the use of a regularization parameter λ. In the easiest form of the model $Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$, the equation to minimize is no longer the least squares equation, but now is:

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

Using calculus, it can be shown that the values for b_i that minimize the above equation are:

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

with n_i as the number of ratings made for movie i. Thus, for a large sample size, the effect of the penalty factor λ is negligible as then $n_i + \lambda \approx n_i$.

As λ is a tuning parameter of the model, I will run a function trying to find the ideal value to minimize the RMSE of our model. In the easiest case of the model with only one factor, this can be done as follows:

```
lambdas <- seq(0, 10, 0.25)
# create function that calculates RMSE for different lambdas
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + l))
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})
# find lambda with the lowest RMSE
lambda <- lambdas[which.min(rmses)]
# add to table
rmse_results <- bind_rows(rmse_results,
                          tibble(model = "Regularized Movie Effect",
                                 RMSE = min(rmses),
                                 difference = min(rmses) - target_rmse))
```

The simplest regularized model now already shows an RMSE value of 0.9442828, highlighting that regularization seems to be a valid way to improve the model without adding more factors. For each one of the models created in chapter 3 so far, I can thus also create a regularized version of the model and add the results to the table. This allows a comparison between each regularized and standard model, thus showing the effect regularization has on improving the RMSE.

```
# create function that calculates RMSE for different lambdas - 2 factor model
rmses_2f <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + l))

  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})
# find lambda with the lowest RMSE
```

```r
lambda_2f <- lambdas[which.min(rmses_2f)]
# add to table
rmse_results <- bind_rows(rmse_results,
                          tibble(model = "Regularized Movie + User Effect",
                                 RMSE = min(rmses_2f),
                                 difference = min(rmses_2f) - target_rmse))

# create function that calculates RMSE for different lambdas - 3 factor model
rmses_3f <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + l))

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

  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_u - b_i - mu)/(n() + l))

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})
# find lambda with the lowest RMSE
lambda_3f <- lambdas[which.min(rmses_3f)]
# add to table
rmse_results <- bind_rows(rmse_results,
                          tibble(model = "Regularized Movie + User + Genre Effect",
                                 RMSE = min(rmses_3f),
                                 difference = min(rmses_3f) - target_rmse))

# create function that calculates RMSE for different lambdas - 4 factor model
rmses_4f <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + l))

  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
```

```
  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_u - b_i - mu)/(n() + l))

  b_rel <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    group_by(RelYear) %>%
    summarize(b_rel = sum(rating - b_u - b_i - b_g - mu)/(n() + l))

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_rel, by = "RelYear") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_rel) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})
# find lambda with the lowest RMSE
lambda_4f <- lambdas[which.min(rmses_4f)]
# add to table
rmse_results <- bind_rows(rmse_results,
                          tibble(model = "Regularized Movie + User + Genre +
                                 Release Year Effect",
                                 RMSE = min(rmses_4f),
                                 difference = min(rmses_4f) - target_rmse))


# create function that calculates RMSE for different lambdas - final model
rmses_final <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + l))

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

  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_u - b_i - mu)/(n() + l))

  b_rel <- train_set %>%
    left_join(b_i, by = "movieId") %>%
```

```
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    group_by(RelYear) %>%
    summarize(b_rel = sum(rating - b_u - b_i - b_g - mu)/(n() + l))

  b_t <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_rel, by = "RelYear") %>%
    group_by(timediff) %>%
    summarize(b_t = sum(rating - b_u - b_i - b_g - b_rel - mu)/(n() + l))

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_rel, by = "RelYear") %>%
    left_join(b_t, by = "timediff") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_rel + b_t) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})
# find lambda with the lowest RMSE
lambda_final <- lambdas[which.min(rmses_final)]
# add to table
rmse_results <- bind_rows(rmse_results,
                          tibble(model = "Regularized Movie + User + Genre +
                                 Release Year + Time Difference Effect",
                                 RMSE = min(rmses_final),
                                 difference = min(rmses_final) - target_rmse))

knitr::kable(rmse_results, caption = "Factor and Regularized Models RMSE")
```

Table 4: Factor and Regularized Models RMSE

model	RMSE	difference
Target RMSE	0.8649000	0.0000000
Basic average	1.0603677	0.1954677
Movie Effect	0.9443770	0.0794770
Movie + User Effect	0.8664927	0.0015927
Movie + User + Genre Effect	0.8661672	0.0012672
Movie + User + Genre + Release Year Effect	0.8660119	0.0011119
Movie + User + Genre + Release Year + Time Difference Effect	0.8656829	0.0007829
Regularized Movie Effect	0.9442828	0.0793828
Regularized Movie + User Effect	0.8657615	0.0008615
Regularized Movie + User + Genre Effect	0.8654748	0.0005748
Regularized Movie + User + Genre + Release Year Effect	0.8653467	0.0004467
Regularized Movie + User + Genre +		

model	RMSE	difference
Release Year + Time Difference Effect	0.8649752	0.0000752

While the final RMSE value of the regularized five-factor model is still higher than the target of RMSE, this is the model I am going to use for the final hold out test against the validation subset that was put aside at the beginning of the project.

The value of lambda for the final hold out test will be 5.

5 Final RMSE Hold Out Test

In this section, I am going to run the model created in chapter 3 on the validation set that was set aside at the beginning of the exercise. As a reminder, the target of this was to achieve a loss as measured by Root Mean Squared Error lower than 0.8649.

```
# set up validation set to feature all new columns from part 3
validation <- validation %>%
  mutate(RevYear = year(as_datetime(timestamp)),
         RelYear = as.numeric(str_sub(title, -5, -2)),
         timediff = ifelse(RevYear-RelYear <0, 0, RevYear - RelYear))

# calculate average rating across edx data set
mu_final <- mean(edx$rating)

# calculate movie factor
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_final)/(n() + lambda_final))
# calculate user factor
b_u <- edx %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_final - b_i)/(n() + lambda_final))
# calculate genre factor
b_g <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu_final - b_i - b_u)/(n() + lambda_final))
# calculate release year factor
b_rel <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  group_by(RelYear) %>%
  summarize(b_rel = sum(rating - mu_final - b_i - b_u - b_g)/(n() + lambda_final))
# calculate time difference factor
b_t <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_rel, by = "RelYear") %>%
  group_by(timediff) %>%
  summarize(b_t = sum(rating - mu_final - b_i - b_u - b_g - b_rel)/(n() + lambda_final))
# calculate predictions for final hold out test
predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_rel, by = "RelYear") %>%
  left_join(b_t, by = "timediff") %>%
  mutate(pred = mu_final + b_i + b_u + b_g + b_rel + b_t) %>%
  pull(pred)
```

```r
# final test
hold_out_rmse <- RMSE(predicted_ratings, validation$rating)
# add to table
rmse_results <- bind_rows(rmse_results,
                          tibble(model = "Final Hold Out Test",
                                 RMSE = hold_out_rmse,
                                 difference = hold_out_rmse - target_rmse))
```

Using the final regularized five-factor model with factors accounting for individual movies and users, movie genres, release years as well as the time between the movie release and eventual rating, I can achieve a loss as measured in RMSE of 0.8639332, which is below the target mark of 0.8649.

Table 5: Final Hold Out RMSE

model	RMSE	difference
Target RMSE	0.8649000	
Final Hold Out Test	0.8639332	-0.000966750743873201

6 Conclusion

While the model I built was able to reduce the Root Mean Squared Error of the predictions to below the target set by the HarvardX course that this report is the capstone project for, a RMSE value of 0.8639332 still seems too high on a scale from zero to five to be called a very good prediction system. Further efforts with advanced techniques such as cross-validation could help improve the predictions even more. This further work on the prediction system could be part of a future project and will not be discussed any more as part of this report.