Capstone Project HarvardX Data Science

Build a Recommendation System

Jan L. Döring

2022-11-03

Contents

Introduction

In 2006, the streaming service Netflix announced a machine learning and data science competition for movie rating prediction. Whoever improved the accuracy of the existing system Cinematch by 10% or more was offered 1 million dollars. By increasing accuracy of the rating prediction system, Netflix could improve recommendations for movies and TV shows which they provide to their users. A well-functioning recommendation system is a key part of the business of streaming services, as it significantly increases the customer satisfaction and loyalty. Therefore, recommendation systems are also widely used in e-commerce and music streaming services. A rating from 1 to 5 stars is the most common scale, where 1 star represents very low customer's satisfaction and 5 stars represents the highest possible customer satisfaction. A simple recommendation system based on ratings can be improved by other predictors, such as search requests on a website, shared links with other users, and user information such as age and sex. In this practical approach, I build a movie recommendation system based on the 10M MovieLens dataset in order to apply lessons taught in the HarvardX Data Science course.

Dataset

The dataset is provided by the GroupLens research lab of the University of Minnesota. It contains 27 million ratings of 58,000 movies by 280,000 users. The data used in this approach is a subset of the MovieLens dataset with 10 million ratings of 10,000 movies rated by 72,000 users.

Model evaluation

Usually in machine learning, the predicted value and the actual outcome are compared. The so-called "loss function" measures the differences between these two values. The most common metrics for recommendation system are the mean absolute error (MAE), the mean squared error (MSE) and the root mean squared error (RMSE). A hypothetical perfect model would return an error of 0, so the lower the value of the given metric the better is, the prediction of the model. The goal of this practical approach is to build a model which returns a RMSE of 0.8649 applied to the 10 M MovieLens subset.

Mean Absolute Error

The mean absolute error represents the average of the absolute distance between the predicted value $\hat{y}_{ui}$ and the actual value y_{ui}. It measures the average of the residuals in the dataset. This error has a linear behavior and makes errors for two different models easily comparable.

$$MAE = \frac{1}{N} \sum_{u,i=1}^{n} |\hat{y}_{ui} - y_{ui}|$$

Mean Squared Error

The Mean Squared Error represents the average of the squared difference between the predicted value and the actual value in the dataset. It measures the variance of the residuals. This error grows exponentially and has a different unit compared to the dependent variable.

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

Root Mean Squared Error

The root mean squared error is the square root of Mean Squared error. It measures the standard deviation of residuals. RMSE is widely used than MSE to evaluate the performance of the model with other random models as it has the same units as the dependent variable

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

Data Science Process

The main steps in a data science project include:

1. Data collection
2. Data preparation
3. Data exploration
4. Data cleaning
5. Data analyses and visualization
6. Model building and validation
7. Creating a report and publishing the results

Methods

Data Collection

GroupLens Research has collected and made available rating datasets from the MovieLens website https://movielens.org. The datasets were collected over various periods of time, depending on the size of the set.

Data preparation

As a first step, I download the data in a compressed folder from the MovieLens website and unzip the folder. The rating data contains userId, movieId, rating, and timestamp. The data is separated by two double dots, so we substitute the dots by tabs. We also add column names for the columns.

The movie data contains the movieId, title, and genres and is provided as a string. We split the string by the two double dots into 3 columns and name the columns. Since both datasets contain the movieId, we can merge the two datasets into one dataset by the left join function and call the dataset movielens.

We split the movielens dataset into two partitions: a training set called edx and contains 90% of the data and a test set called validation with the remaining 10% of the data.

```
dim(edx_raw)
```

```
## [1] 9000047       6
```

```
dim(validation_raw)
```

```
## [1] 1000007       6
```

In order to build a model that delivers a RMSE lower than the value of the project goal, we again split the edx partition in two smaller partitions where 90% represent the training set and 10% the test set. If we reach our project goal with these partitions, we train the model again with the entire edx dataset and validate the model with the validation dataset. This holdout method is a type of cross validation and is executed to get an impression how our models are doing on unseen datasets.

Data Exploration

As a first step,it is important to get an idea of the structure of our dataset. For this we can call the str() function to give us information about the variables and the datatype.

```
str(edx_raw)
```

```
## Classes 'data.table' and 'data.frame':   9000047 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
##  - attr(*, ".internal.selfref")=<externalptr>
```

We can observe that our dataset is of the type data.frame with 900055 rows and 6 columns. The columns have different datatypes:

movieId - integer
userId - integer
ratic - numeric
timestamp - numeric
title - characterstring
genres - chracterstring

With the head() function we can call the first 6 entries of the dataset. We observe that the data.frame is in a tidy format meaning that each row represents a single observation while the columns represent the features of this observation. The column rating is the value that we want to predict with our future model.

```
head(edx_raw)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

We can observe that our dataset has a timestamp column, making the interpretation of the date of rating quite hard. Another fact that we can observe is that the release date is included in the title as characterstring. The genre column does not consist of only one single genre, rather it is often a combination of different genres to which the film can be assigned.

Another important thing with a huge dataset like this is to make sure that we don't have any missing values (NA) or values that equal zero. So we perform a check on NA and zeros like this. If we'd have any NA in our dataset the average of the returned values of the function is.na() applied to the edx dataset should be > 0.

```
mean(is.na(edx_raw))
```

The function below tests if our dataset has any zeros. It returns FALSE if there is no zeros in the dataset and TRUE if there are any zeros.

```
all(apply(apply(edx_raw, 2, function(x) x==0), 2, any))
```

Since the average of the values produced by the is.na() function applied to our edx dataset is 0, we can be sure that we don't have to deal with any missing values. The nested applied function which returns FALSE tells us that we don't have any values which equal 0. This means that there is also no rating with the value of 0.

In order to convert the timestamp into a date with the format YY-MM-DD, we execute the following code:

```
edx$date <- as.POSIXct(edx$timestamp, origin="1970-01-01")
validation$date <- as.POSIXct(validation$timestamp, origin="1970-01-01")

edx$year_of_rating <- format(edx$date,"%Y")
edx$month_of_rating <- format(edx$date,"%m")
validation$year_of_rating <- format(validation$date,"%Y")
validation$month_of_rating <- format(validation$date,"%m")
```

Like this we not only convert the date, we also extract the year and month of the rating in order to find some possible relations to rating patterns at certain times.

If we also want to determine the relationship between the rating and the release date, we have to extract the release date from the title. For this we use the the following code:

```
edx_release <- stringi::stri_extract(edx$title, regex = "(\\(\\d{4}\\))", comments = TRUE)
edx_release <- gsub("[()]", "", edx_release) %>% as.numeric()
edx <- edx %>% mutate(release_date = edx_release)

validation_release <- stringi::stri_extract(validation$title, regex = "(\\(\\d{4}\\))", comments = TRUE
validation_release <- gsub("[()]", "", validation_release) %>% as.numeric()
validation <- validation %>% mutate(release_date = validation_release)
```

The regular expression used in the stringi::string_extract() function extracts any digits in parentheses. We do it like this since we want to avoid to extract any numbers which might be part of the title. Afterwards, we remove the parentheses with the gsub() function and cast the string digits to numeric data type format.

To separate the genres column, we use the separate_rows() function to separate the string by the special character (\|). This enlarges the dataset a lot since every rating gets assigned to every single genre that the movie is assigned to.

```
edx_genre_sep <- edx %>%
  mutate(genre = fct_explicit_na(genres, na_level = "(not assigned to a genre)")) %>%
  separate_rows(genre, sep = "\\|")
```

Data Cleaning

In order to remove unnecessary columns, we just select the desired columns and save them to the dataset:

```
edx <- edx %>%
  select(userId, movieId, rating, title, genres, release_date, year_of_rating, month_of_rating)

validation <- validation %>%
  select(userId, movieId, rating, title, genres, release_date, year_of_rating, month_of_rating)
```

As a last step we convert the data types of the year_of_rating and month_of_rating columns into a numeric data type:

```
edx$year_of_rating <- as.numeric(edx$year_of_rating)
edx$month_of_rating <- as.numeric(edx$month_of_rating)

validation$year_of_rating <- as.numeric(validation$year_of_rating)
validation$month_of_rasting <- as.numeric(validation$month_of_rating)
```

Now that we have prepared our datasets into the desired shape, we can execute another split of the edx dataset into to partitions, a training set and test set:

```
dim(edx)
```

```
## [1] 9000055       8
```

```
dim(test_set)
```

```
## [1] 899990      8
```

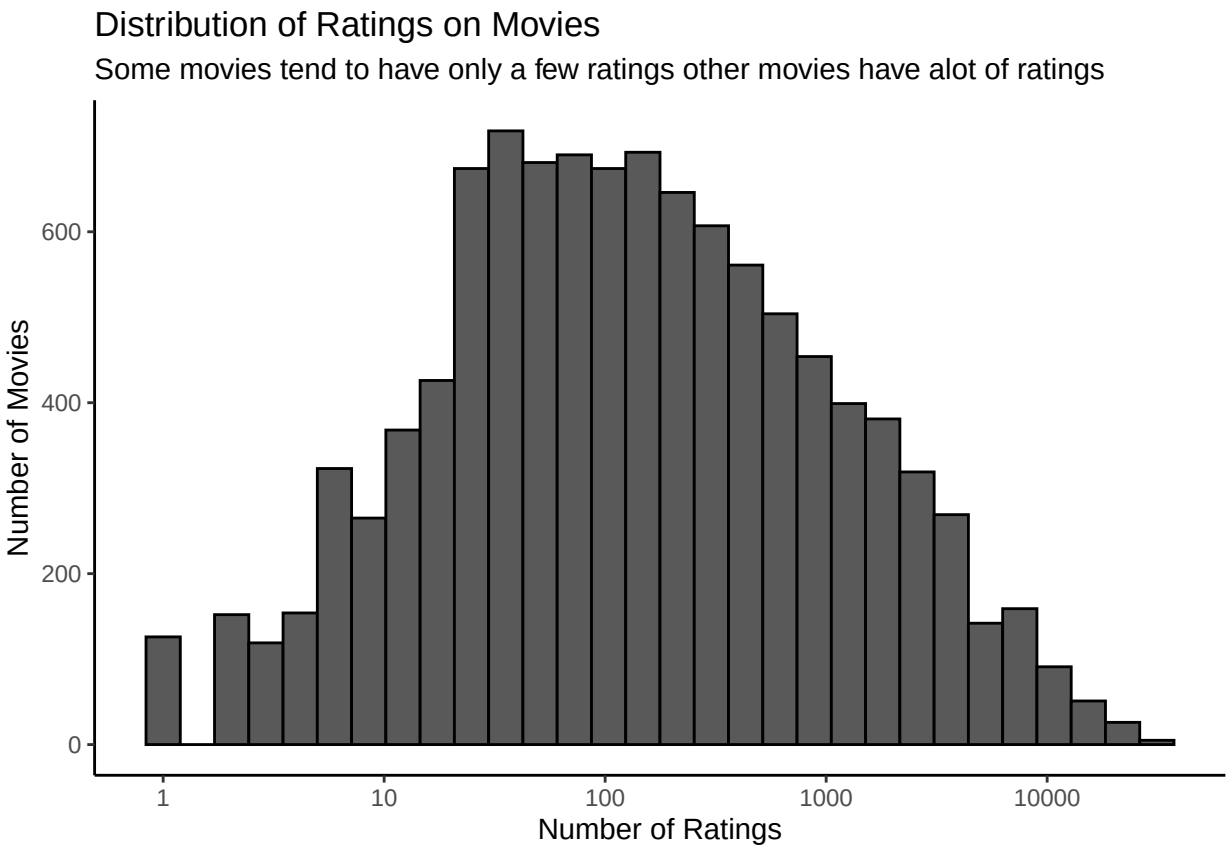
```
dim(train_set)
```

```
## [1] 8100065       8
```

Data Aanalysis and Visualization

Movies

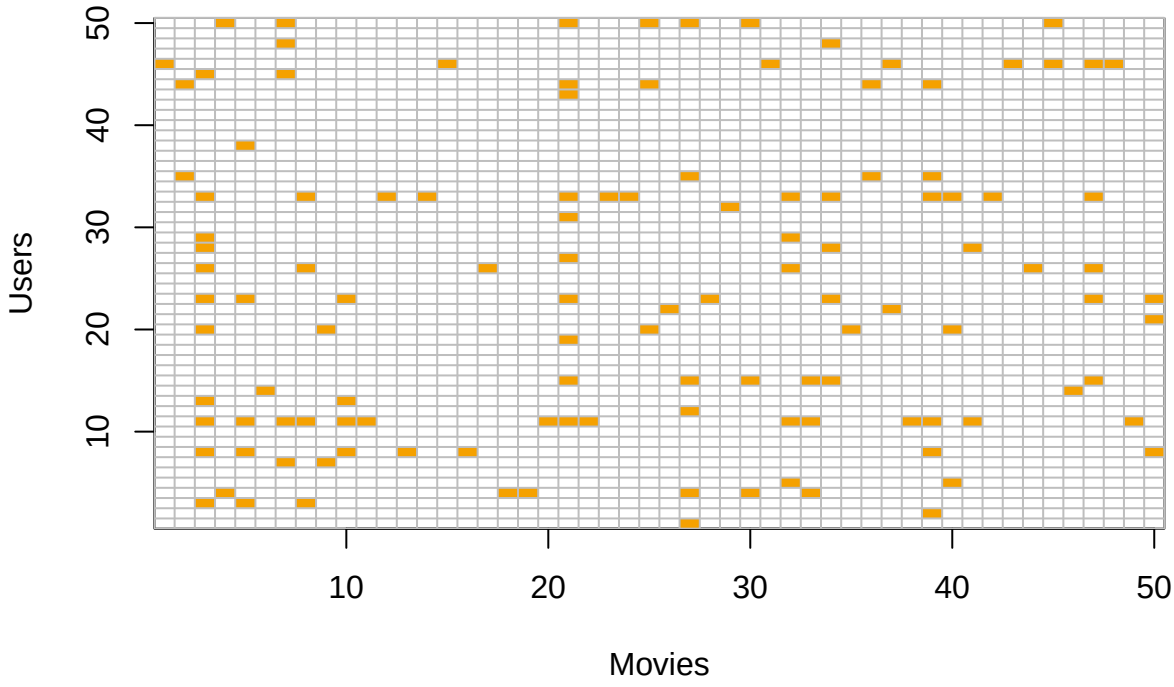
In this section we explore the distribution of ratings and movies. We want to answer the question if movies tend to have the same amount of ratings. For this we plot the number of movies versus the number of ratings:



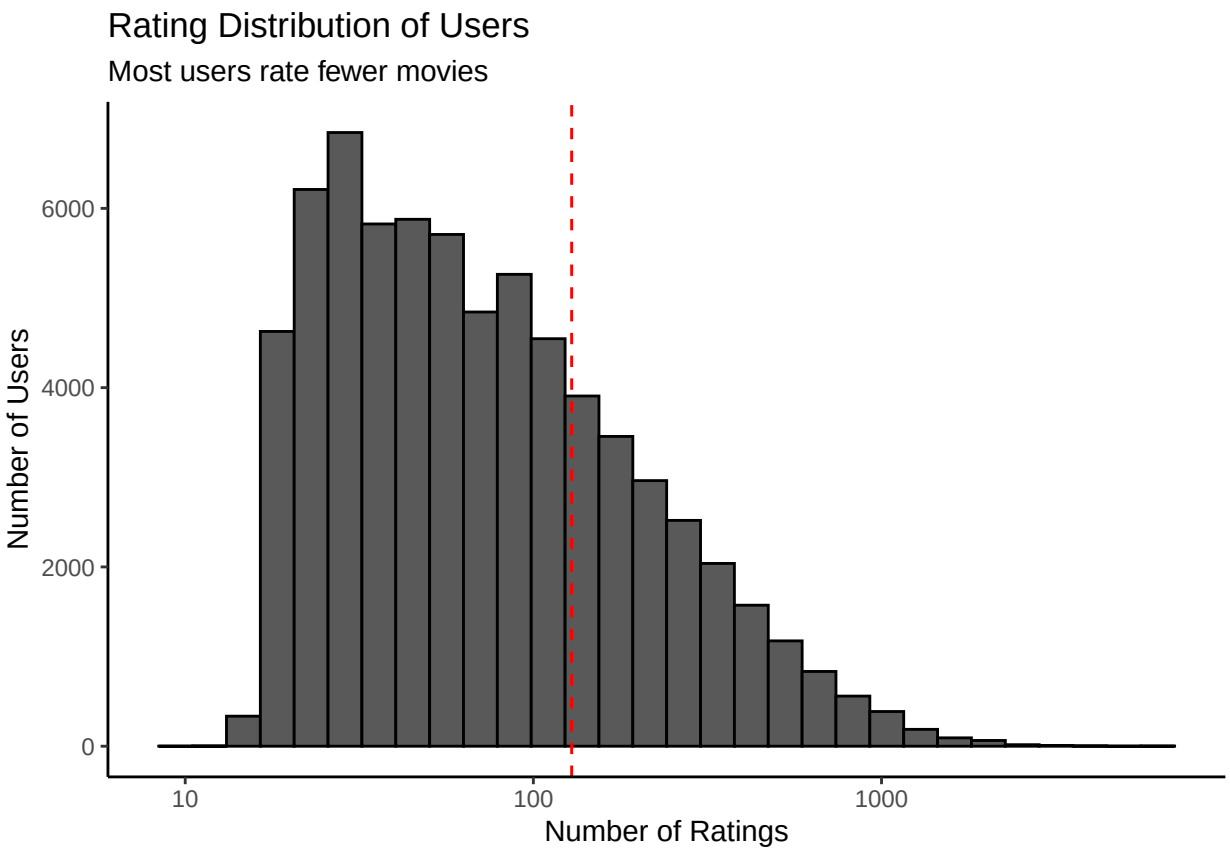
It turns out that there is a distribution of ratings on movies which has approximately a symmetric shape. We can see that there is some movies which have only a few ratings while there is some others which got rated very often.

Users

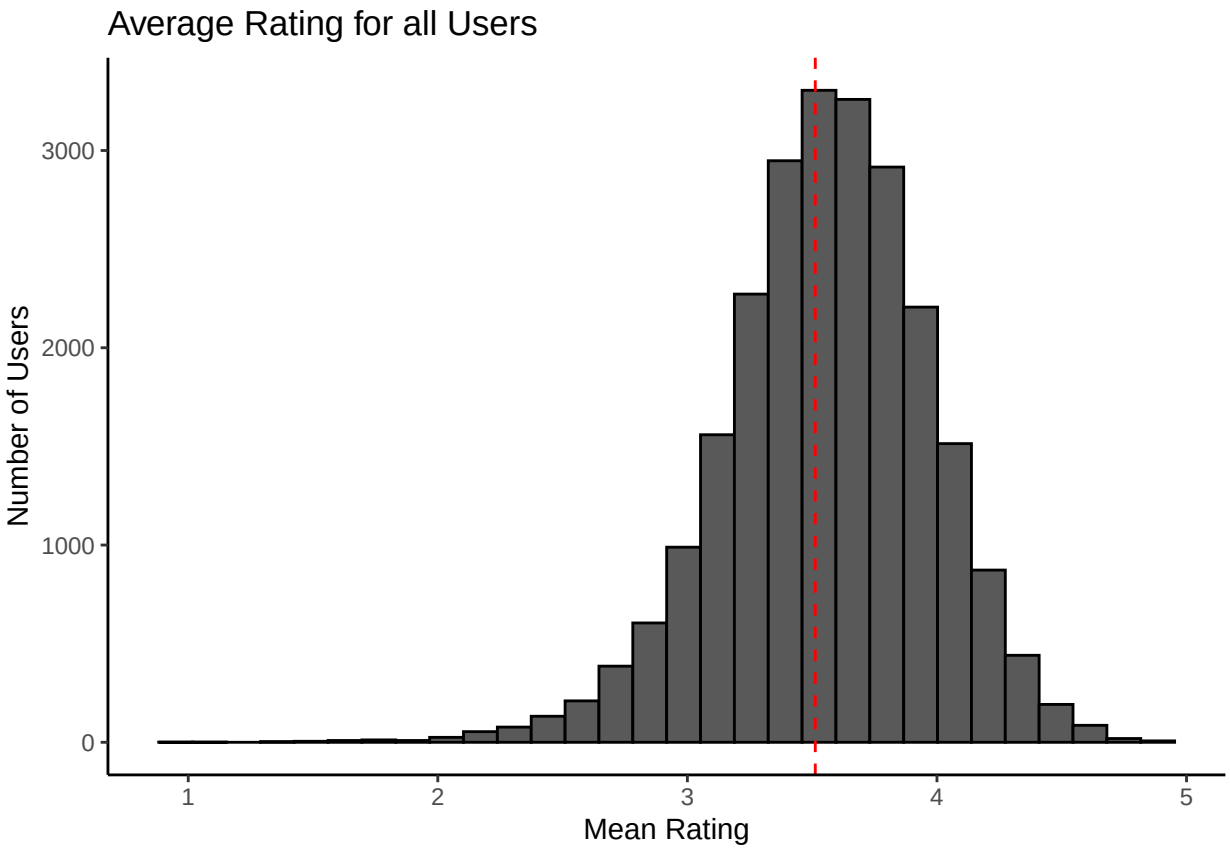
In order to show how sparse the entire M x U matrix is, we plot a smaller subset of 50 movies and 50 users to get an idea how many users have rated a movie. The yellow tiles represent a rating.



The number of ratings for users is very different as we can see in the next plot. While some users rated thousands of movies most of the other users only rate a small amount. The red dashed line represents the average user which rates 129 movies.

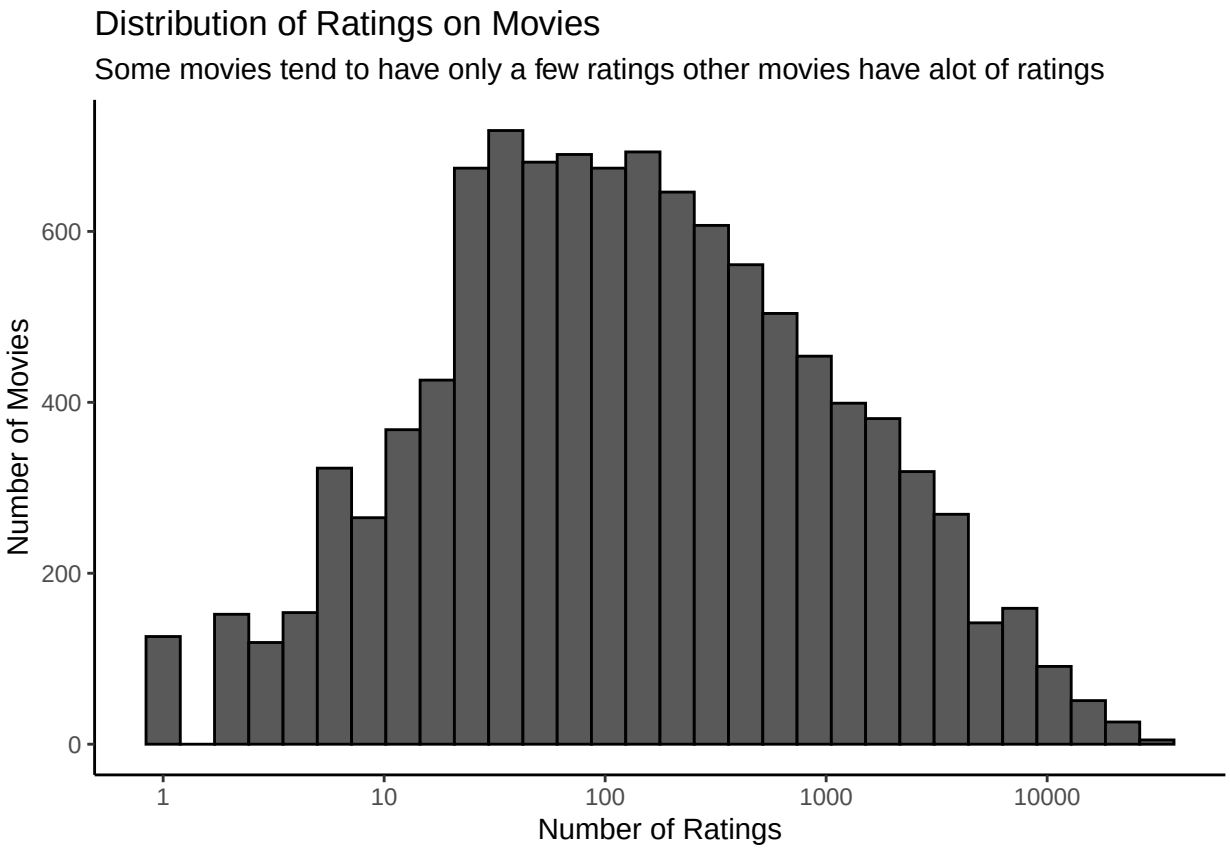


We can also investigate if there is a user effect depending on the mean rating per user. For this we plot a histogram of the average rating of users with more than 100 reviews. The red dashed line represent the overall average rating.



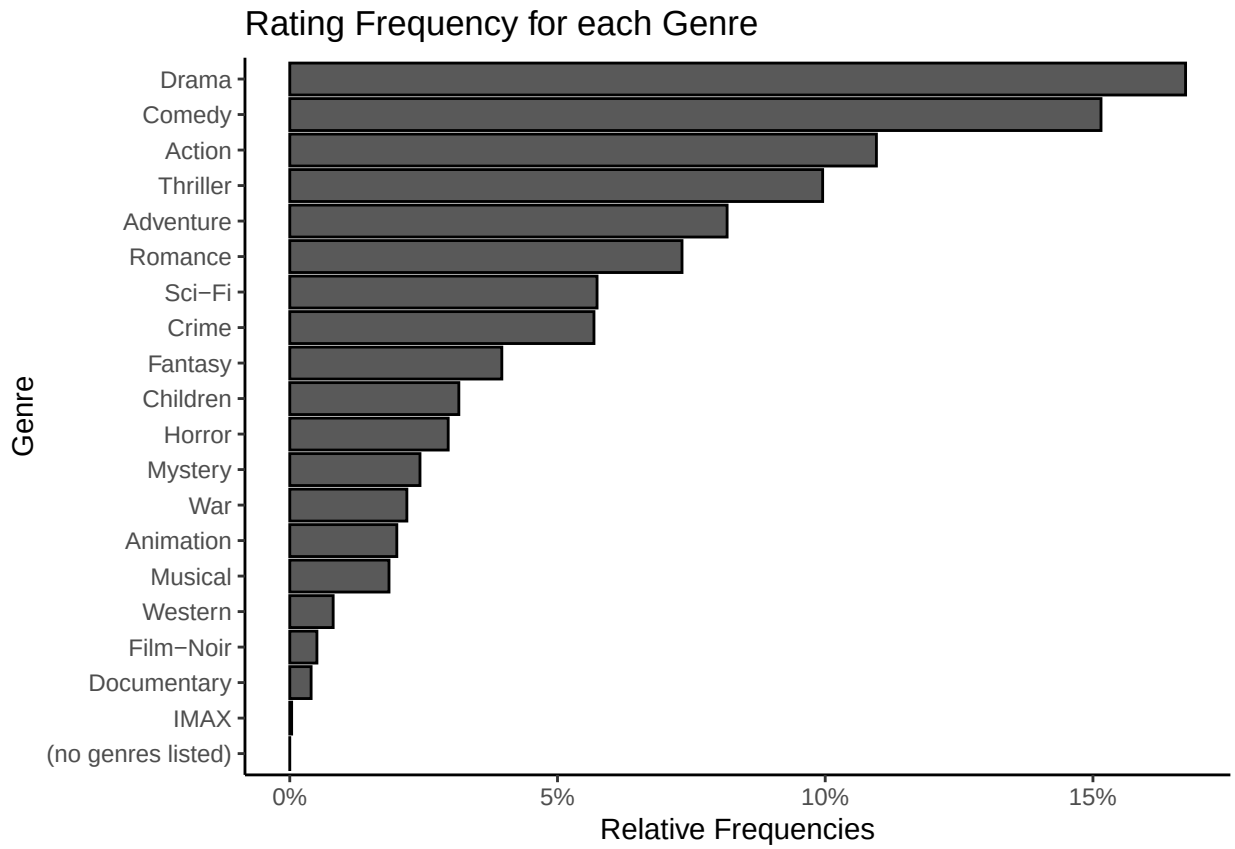
Ratings

The ratings in the dataset are numeric values from 0.5 to 5.0. Although one might think that even ratings and odd ratings occur in the same frequency the opposite is the case. Most people tend to give even ratings on movies. We already calculated the mean rating so that we know the average rating of all movies is approximately 3.51. The following plot visualizes this fact that people give higher ratings more often than lower ratings.

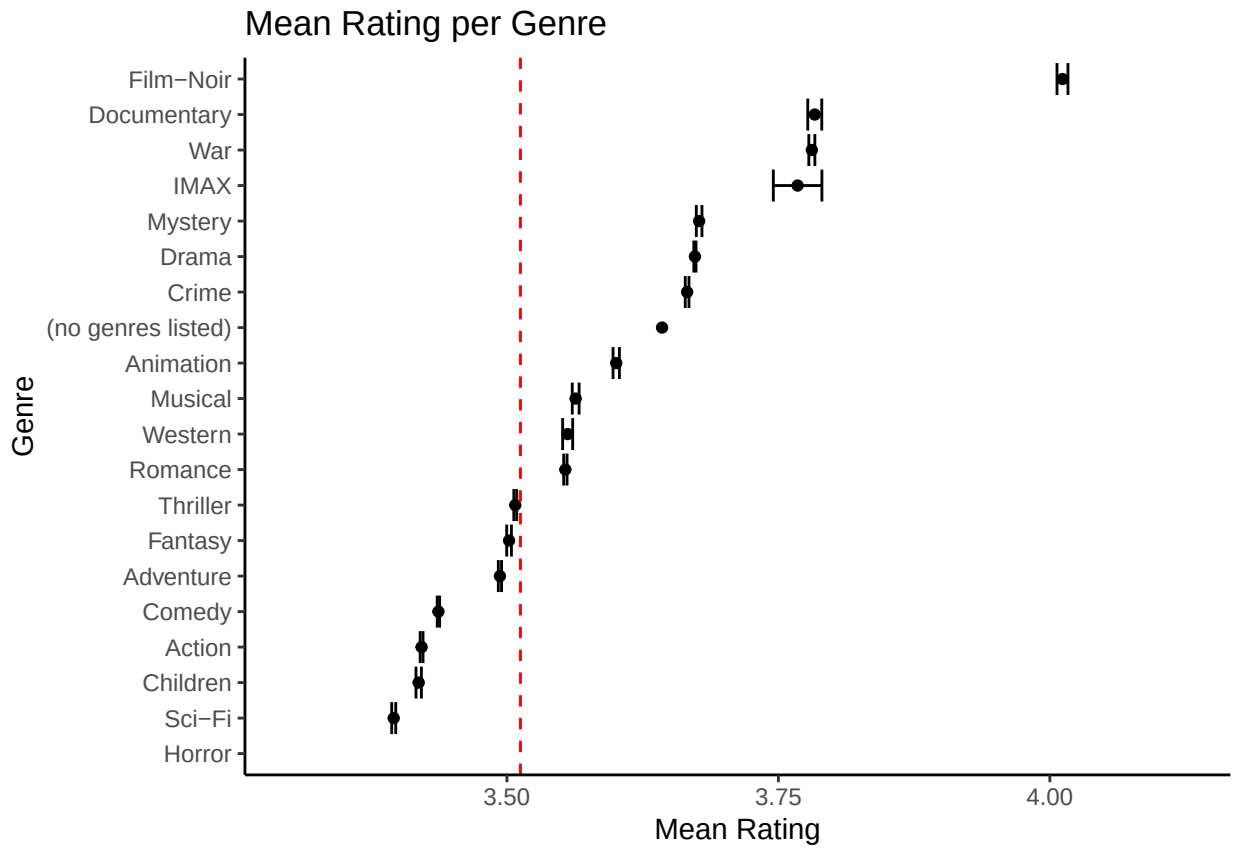


Genre

The genres column in the dataset is most of the time a combination of different genres. With a separation function we managed to split the column into several columns in order to plot the relative frequency of a single genre to be assigned to a movie.

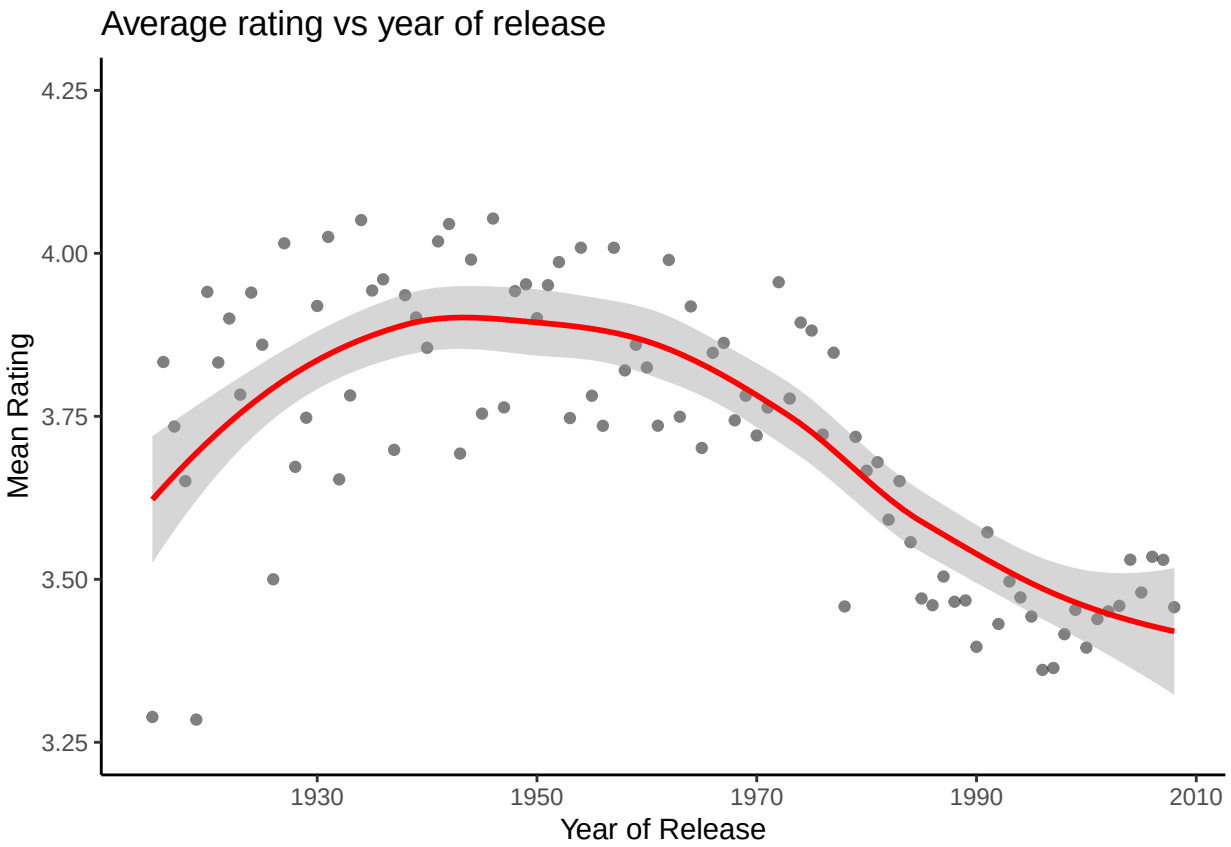


To investigate if the rating depends on the genre the average ratings are analyzed per genre.



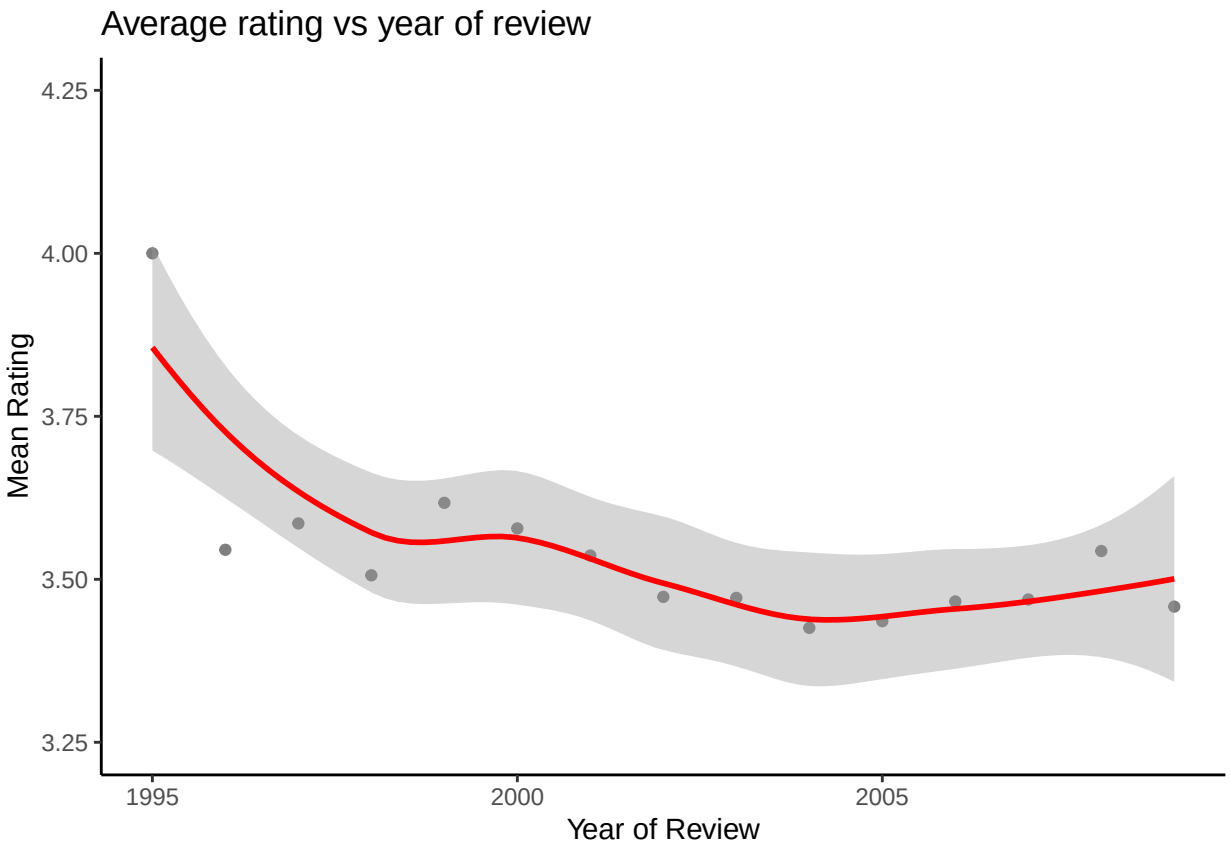
Date

The release date of the movie might also have an impact on the mean rating for that movie. In order to find tendencies we plot the mean rating versus the release year.



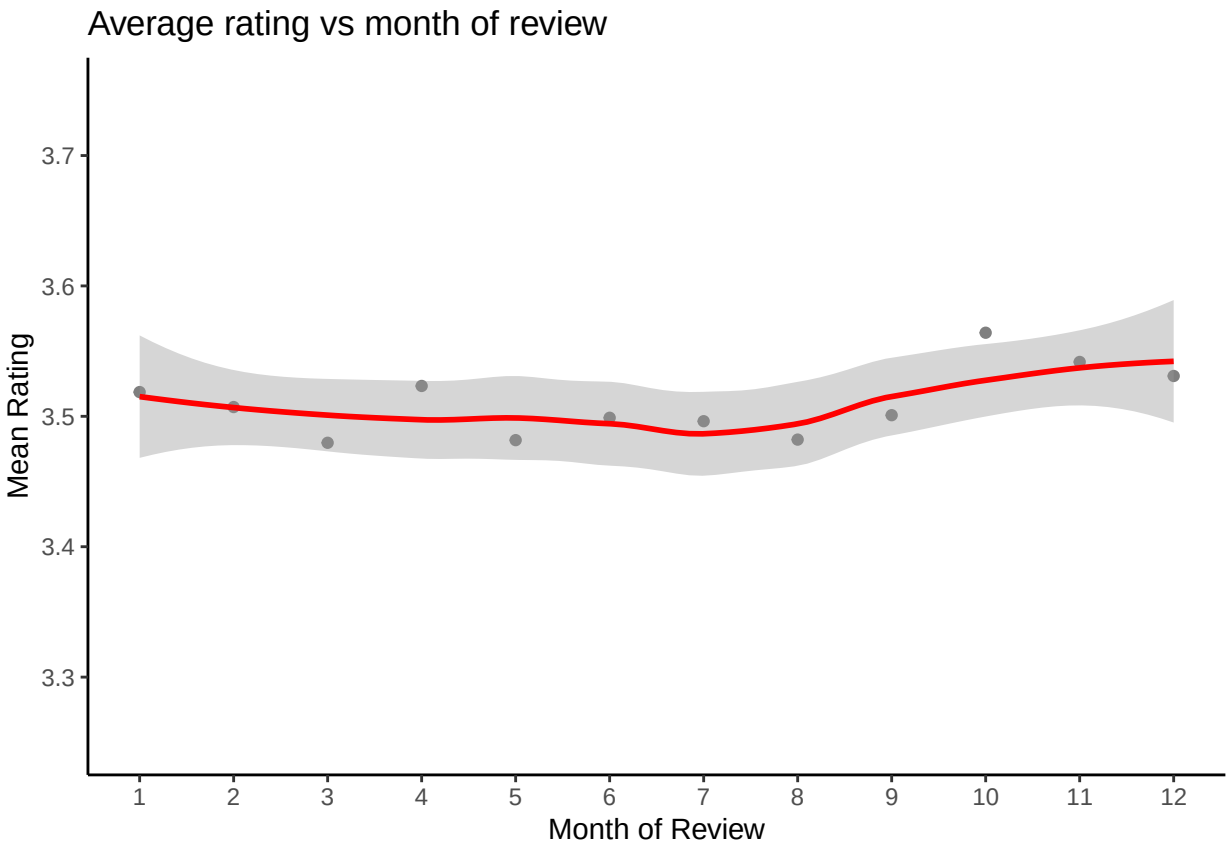
We can observe that most newer movies have a lower average rating than older movies, while the variance of the average ratings of older movies is higher. So we can assume that there might be a small time effect on movie ratings, although it might be not affect the rating so much.

With the next plot we want to figure out if there is an time effect depending on the year of the review.



we can see that there is some higher ratings in year 1995 which lead to an average rating of 4 Stars in this years. But it seems to be an outlier compared to the other values which are quite on the same level between 3.4 and 3.7.

With the next plot we want to find out if the month of rating somehow has an impact on the average rating during this month.



It turns out that there is no time effect based on the date of review. The averages seem to differ only marginally.

Model Building

Random Prediction

The most simple approach is to just randomly predict the rating by the probability distribution observed in Figure 1. We can run a Monte Carlo simulation and estimate the probability on each rating so that we can predict random ratings with sampling.

Linear Model

A simple linear model can be build based on the assumption, that all users give same ratings on the same movies. In this model each differences are explained by random variation. The true rating for all movies and all users is represented by μ and ϵ is the independent error with same distribution and a center at zero, the following formula gives the prediction:

$$\hat{y}_{ui} = \mu + \epsilon_{i,u}$$

The movie to movie variability can be explained by the fact that different movies have different rating distributions. Some movies are more popular than other movies so they have more ratings. We can improve our model if we add a term b_i which represents the average rating of movie i and we call it the movie effect.

$$\hat{y}_{i,u} = \mu + \epsilon_{i,u} + b_i$$

The movie effect b_i can be estimated by calculating the mean of rating y_{ui} minus the overall mean.

$$\hat{b}_i = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{\mu})$$

Considering that users have different rating patterns, we might be able to build a better model. Some users tend to rate every movie they watched very high, while others rate every movie very low. So a rating of where user A who generally rates movies very low gives 3 stars is a much better rating compared to user B who only gives high ratings and also gave 3 stars. We add a term b_u which we call the user specific effect.

$$\hat{y}_{i,u} = \mu + \epsilon_{i,u} + b_i + b_u$$

The user effect can be calculated by the the mean of rating y_{ui} minus the overall mean minus the overall mean and minus b_i.

$$\hat{b}_u = \frac{1}{N}\sum_{i=1}^{N}(y_{i,u} - \hat{\mu} - \hat{b}_i)$$

The genre effect can be calculated like this:

$$\hat{y}_{i,u,g} = \mu + \epsilon_{i,u} + b_i + b_u + b_g$$

$$\hat{b}_g = \frac{1}{N}\sum_{i=1}^{N}(y_{i,u} - \hat{\mu} - \hat{b}_i - \hat{b}_u)$$

One can also observe that the release date affects the rating. Older movies have higher average ratings but also more variability, newer movies tend to have a slightly lower average rating with less variability. We add the term b_t which we call the release date affect to our linear model in order to improve our predictions.

$$\hat{y}_{i,u,g,t} = \mu + \epsilon_{i,u} + b_i + b_u + b_g + b_t$$

The release date effect can be calculated like this:

$$\hat{b}_t = \frac{1}{N}\sum_{i=1}^{N}(y_{i,u,g} - \hat{\mu} - \hat{b}_i - \hat{b}_u - \hat{b}_g)$$

Regularization

Although a linear model like this can give good predictions, it doesn't take into a account that some movies have only a very small amount of ratings, some users only give a very small amount of ratings and some years have less released movies than others. In statistics a small sample size generally leads to larger errors. So for users, movies and release years we want to add a factor that penalizes a small sample size. We add the term λ in our calculations for movie, user and release date effects so that the impact on our predictions get reduced.

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

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

$$\hat{b}_g(\lambda) = \frac{1}{\lambda + n_g} \sum_{i,u=1}^{n_g} (y_{i,u,g} - \hat{\mu} - \hat{b}_i - \hat{b}_u)$$

$$\hat{b}_t(\lambda) = \frac{1}{\lambda + n_t} \sum_{i,u,g=1}^{n_t} (y_{i,u,g,t} - \hat{\mu} - \hat{b}_i - \hat{b}_u - \hat{b}_g)$$

This penalization factor λ reduces the values of $\hat{b}_i$, $\hat{b}_u$ and $\hat{b}_t$ when the number of ratings n_i, n_u, n_t is very small. These values don't change so much if the number of ratings is very high. We can think of lambda being a tuning parameter so we will use different values of λ in order to find the λ that gives us the least root mean squared error (RMSE).

Matrix Facorization

Matrix Factorization is a very common machine learning method for recommendation systems. As a first step the data gets converted into a User x Movie matrix, where each row represents a User and each column represents a Movie. We already saw in the sampled sparse matrix, that such a matrix has a lot of missing values since users never saw and rate all of the existing movies. The algorithm attempts to fill the empty slots with predictions how these users would rate the movie. The idea of Matrix Factorization is to find the missing values based on hidden features that we don't know and also can't weight properly. Some hidden features could be title, main actors, genre, script authors, directors, movie length etc. that users prefer and have an impact on their rating.

A matrix $R = |UxM|$ can be seen as the matrix that contains all the ratings for each user on each movie and the missing values in case that the movie is not rated yet.

In order to discover K features we want to factorize the matrix into $PxQ^T = R$
P has the size $|U|x|K|$ and Q^T has the size $|M|x|K|$.
The product of these two matrices is an approximation of our matrix R.

$$R \approx PxQ^T$$

Let p_u be the uth row of P and q_v the vth row of Q than the rating of user u on movie v should be predicted as $p_u q_v^T$. A solution for P and Q is given by the following optimization problem:

$$\min P, Q \sum_{u,v \in R} ((r_{u,v} - p_u^T q_v)^2 + \lambda_P ||p_u||^2 + \lambda_Q ||q_v||^2)$$

In this formula u, v are the locations of observed entries in R, $r_{u,v}$ is the observed rating, and λ_P,λ_Q are penalty parameters to avoid overfitting.

One of the most popular methods to solve this problem is called stochastic gradient descend (SGD). The basic idea of SGD is to randomly select a entry (u, v) from the summation and calculating the corresponding gradient instead of calculating the gradient of the formula given above. Once r_{uv} is chosen the function is reduced to:

$$(r_{u,v} - p_u^T q_v)^2 + \lambda_P p_u p_u^T + \lambda_Q q_v q_v^T$$

After calculating the sub-gradient of p_u and q_v the variables get update by the following rule:

$$p_u \leftarrow p_u + \gamma(e_{u,v} q_v - \lambda_P p_u)$$
$$q_v \leftarrow q_v + \gamma(e_{u,v} p_u - \lambda_Q q_v)$$

The learning rate γ determines how big the steps for the updated values are so a learning rate $\gamma = 0.1$ will result in bigger steps than a rate of $\gamma = 0.01$.

The error $e_{u,v}$ is calculated by the following formula:

$$e_{u,v} = r_{u,v} - p_u^T q_v$$

The general procedure of SGD is to iteratively select an instance $r_{u,v}$, apply the update rules given above in order to receive smaller errors.

The process of solving the matrices P and Q is referred to as model training, and the selection of penalty parameters is called parameter tuning. A library that provides an algorithm to solve the above problem is provided by LIBMF, a high performance library for large scale matrix factorization. The R library Recosystem is a wrapper of LIBMF and will be used in this project. There is several tuning parameters that we change in order to get better results:

dim -> dimensions (k latent features)
lrate -> learning rate
costp_l2 -> regularization parameter for user factors
costq_l2 -> regularization parameter for item factors
nthread -> number of threads in order to execute multi-threading
niter -> number of iterations

Results

Model evaluation

The models discussed in the Methods section will be applied to the train set and test set which are two partitions of the original edx set. In order to determine the quality of our model we define the loss functions MAE, MSE and RMSE. Only after a achieving a RMSE value lower than our project goal we will apply our final models to our actual train set (edx) and our actual test set (validation).

```
# Mean Absolute Error

MAE <- function(predicted_rating, true_rating){
  mean(abs(predicted_rating - true_rating))
}

# Mean Squared Error

MSE <- function(predicted_rating, true_rating){
  mean((predicted_rating - true_rating)^2)
}

# Root Mean Squared Error

RMSE <- function(predicted_rating, true_rating){
  sqrt(mean((predicted_rating - true_rating)^2))
}
```

Random Prediction Model

Our first model provides a random prediction of ratings using the probabilities for a rating which were estimated with our training set. Afterwards we feed the estimated value and the real value to our loss function in order to calculate the errors.

Comparison of Results
Project Goal: RMSE = 0.8649

Approach	RMSE	MSE	MAE
Random prediction	1.4998	2.2493	1.1664

As expected the value of the RMSE is pretty high compared to our project goal.

Linear Model

The linear model was adapted step by step with the intention to reduce the RMSE further and further. For this purpose, various effects were taken into account.

Mean calculation

As a first simple prediction we just calculated the mean of the ratings:

$$\hat{y}_{ui} = \mu + \epsilon_{i,u}$$

Comparison of Results
Project Goal: RMSE = 0.8649

Approach	RMSE	MSE	MAE
LM mean	1.0601	1.1237	0.8552

The RMSE for this prediction is better than our random prediction since the average minimizes the root mean squared error.

Movie effect

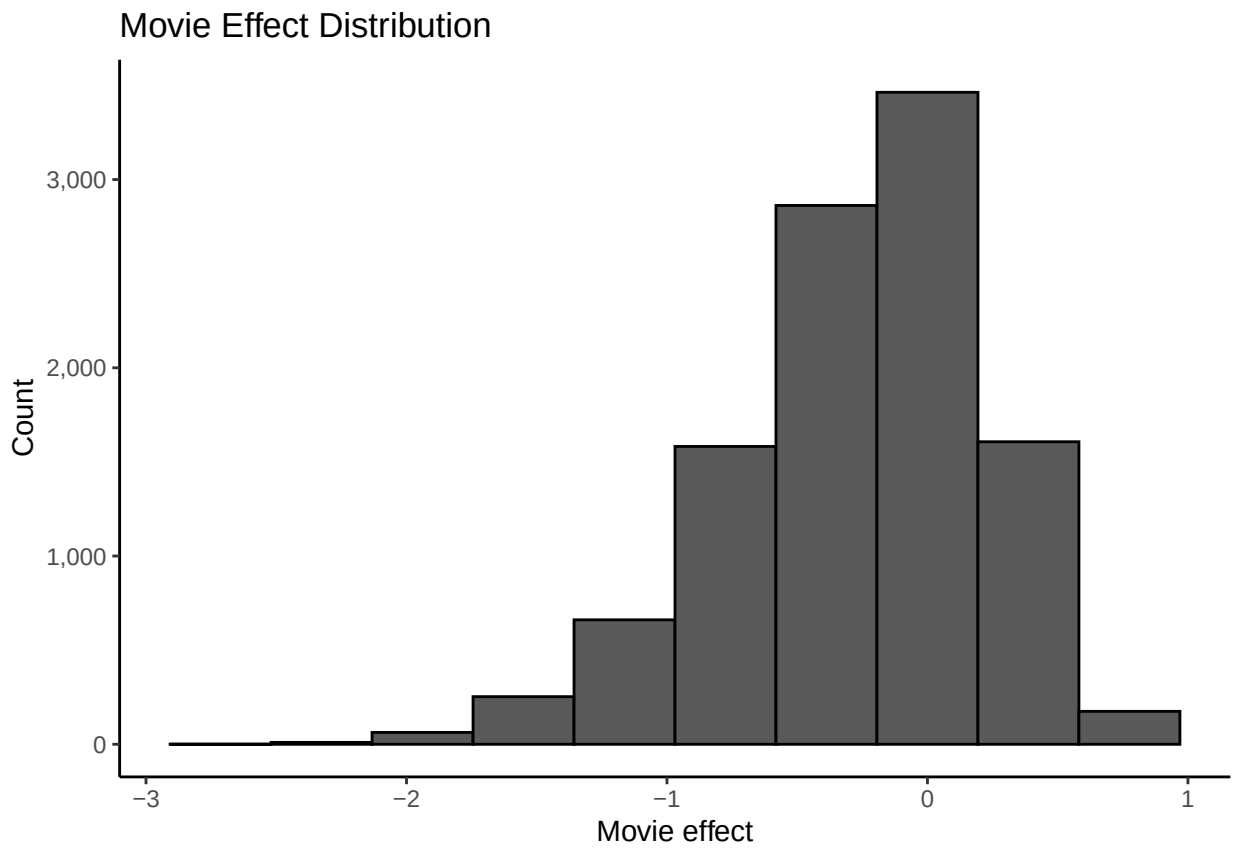
In the next step we included the movie effect:

$$\hat{y}_{i,u} = \mu + \epsilon_{i,u} + b_i$$

Comparison of Results
Project Goal: RMSE = 0.8649

Approach	RMSE	MSE	MAE
LM incl. movie effect	0.9430	0.8892	0.7370

The result is already much better than the prediction by the average rating. With this approach we could lower our RMSE by more than 10%.



The distribution for the values of b_i is not symmetric. Negative values are much more common than postive values.

User effect

In the next step we also included a user specific effect:

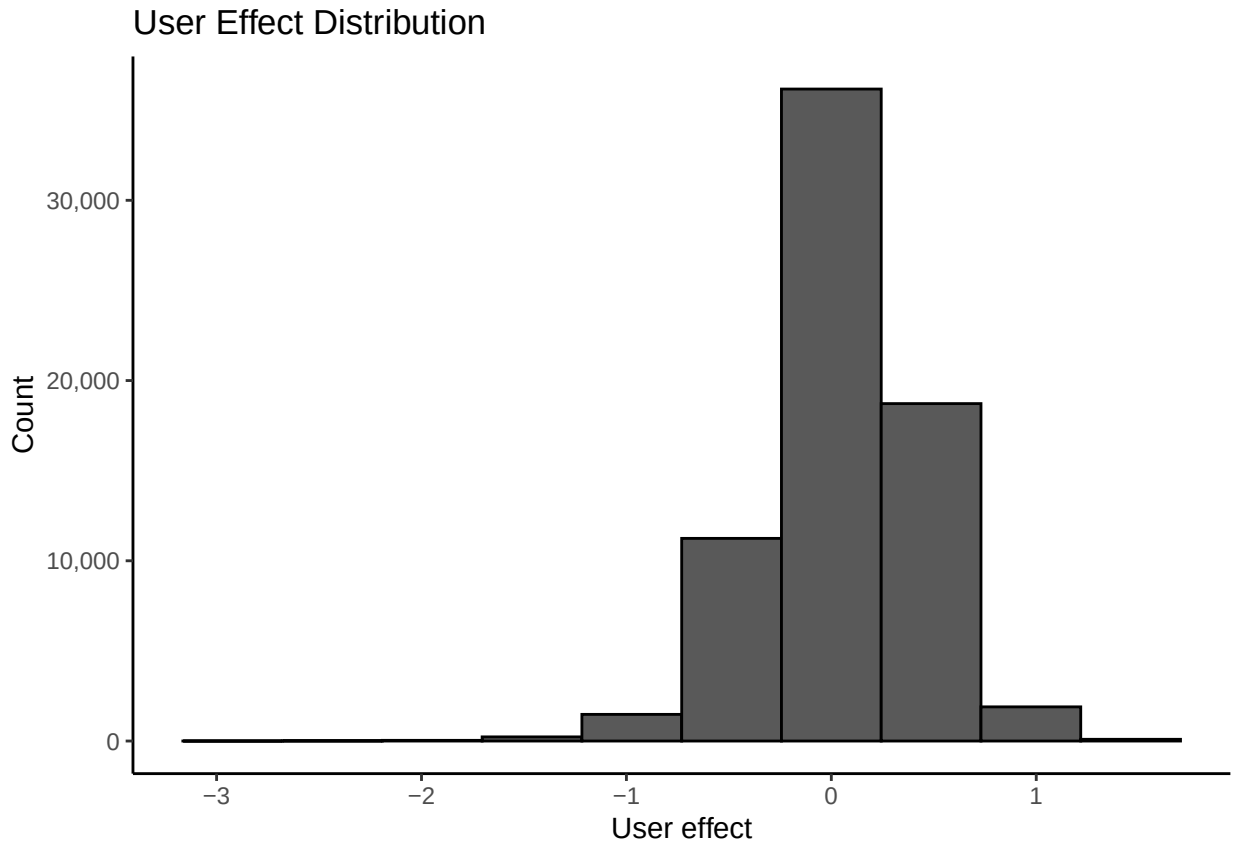
$$\hat{y}_{i,u} = \mu + \epsilon_{i,u} + b_i + b_u$$

Comparison of Results

Project Goal: RMSE = 0.8649

Approach	RMSE	MSE	MAE
LM incl. user effect	0.8647	0.7477	0.6684

Including the user specific effect we could further lower our RMSE by another 8% compared to the last step.



The distribution of the user effects is almost symmetric with the center of 0. There is some more positive values for b_u then negative values.

Genre effect

The next approach also takes a genre effect into account:

$$\hat{y}_{i,u,g} = \mu + \epsilon_{i,u} + b_i + b_u + b_g$$

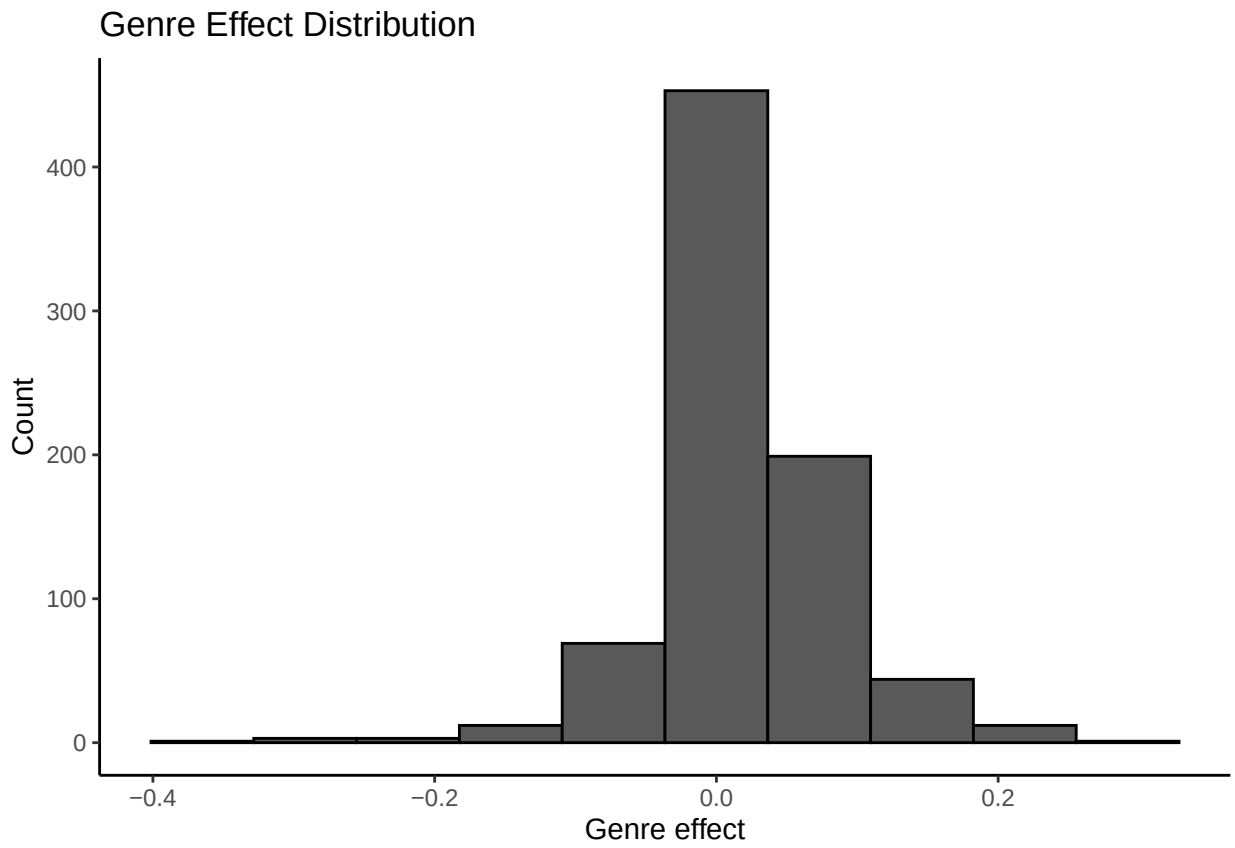
Comparison of Results

Project Goal: RMSE = 0.8649

Approach	RMSE	MSE	MAE

LM incl. genre effect	0.8643	0.7471	0.6679

With this method, the resulting RMSE changes only at the third decimal place compared to our last method. Taking this effect into account may not have such a big impact, but we can still make our model a little better.



The distribution of the genre effect is almost symmetric. The calculated values for b_g are ten times smaller than the the values for b_i and b_u. There are slightly more positive values than negative values.

Release Date effect

As a last step we also added a release date effect.

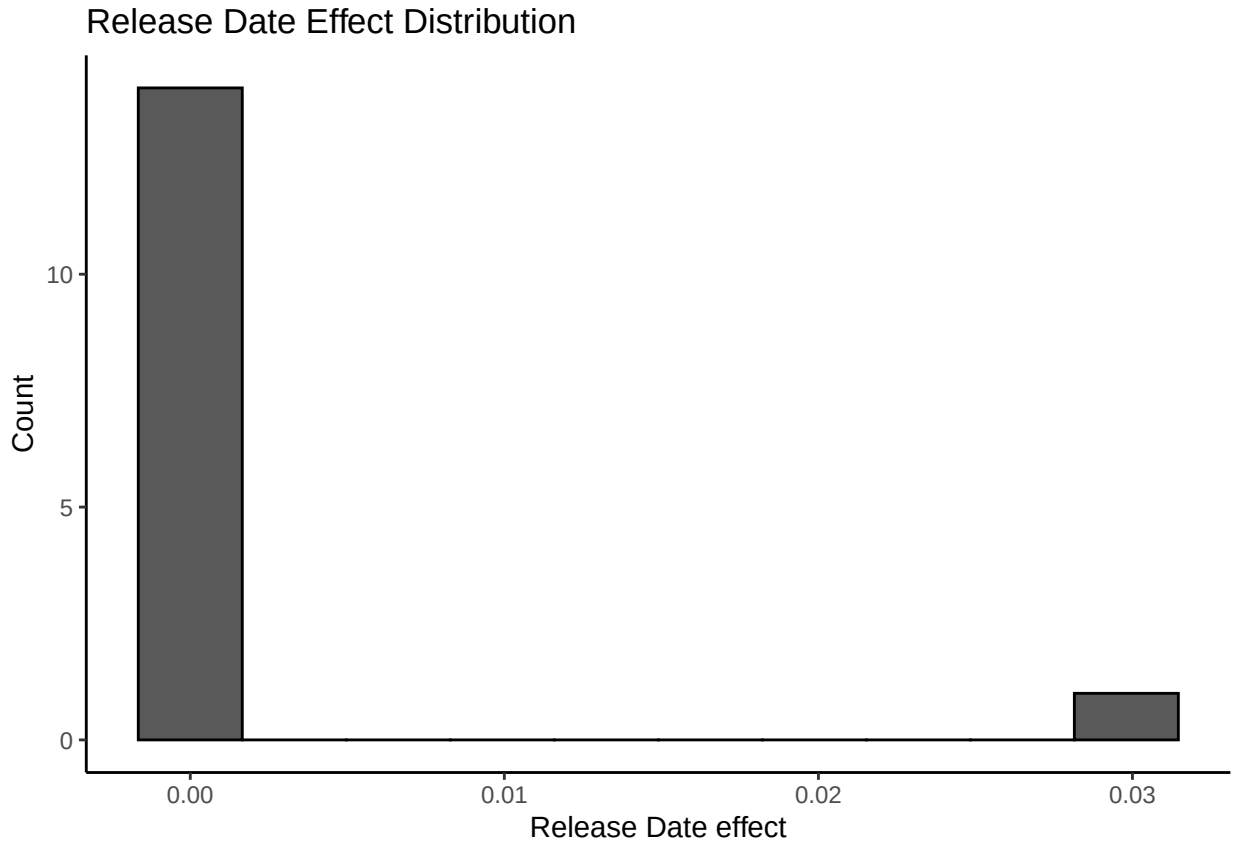
$$\hat{y}_{i,u,g,t} = \mu + \epsilon_{i,u} + b_i + b_u + b_g + b_t$$

Comparison of Results

Project Goal: RMSE = 0.8649

Approach	RMSE	MSE	MAE
LM incl. timeffect	0.8643	0.7471	0.6679

This addition of a time based effect doesn't seem to have any impact on our result. The RMSE has the same value as our model only based on movie, user and genre effect.



We can see that the nearly all biases have a value of zero which explains that there is no effect on our model. Only one outlier is observable but the value is 100 times smaller than the values of b_i and b_u.

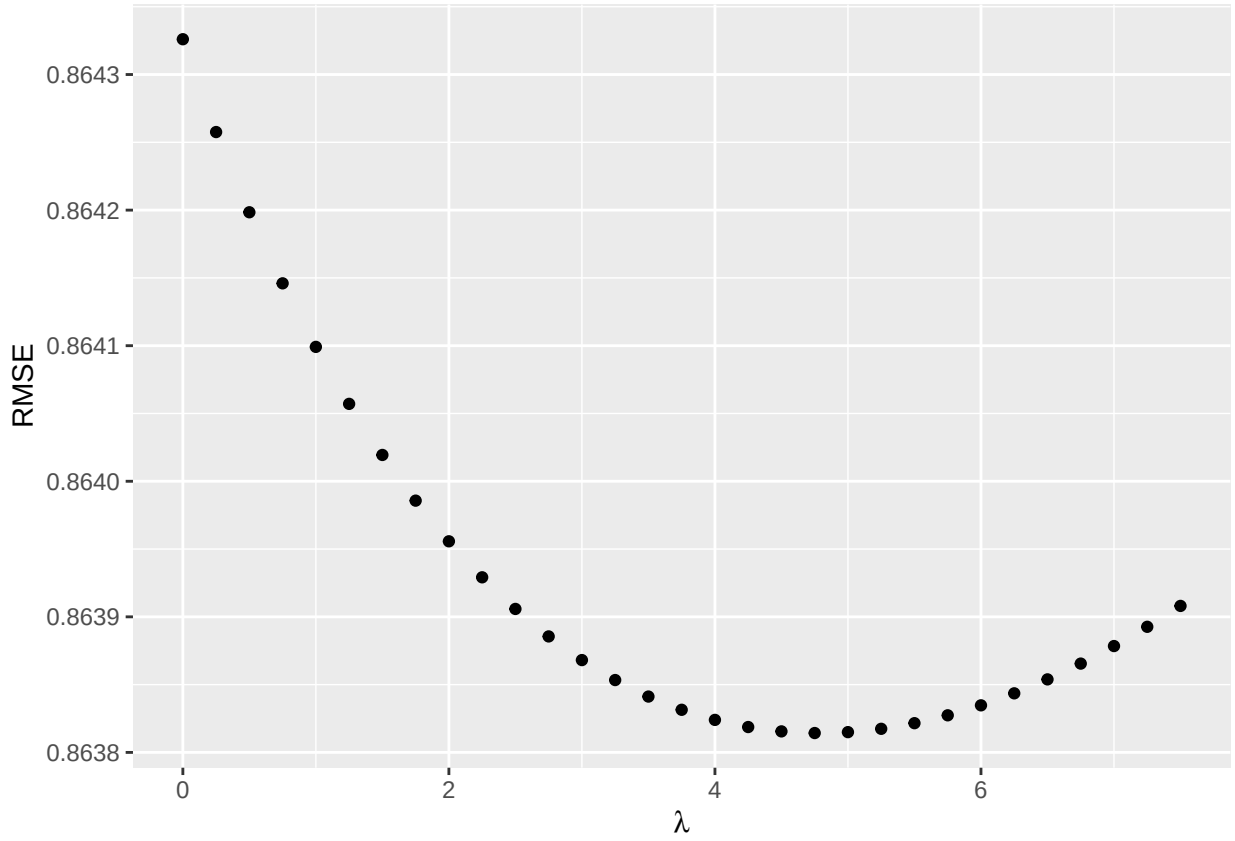
Regularized Linear Model

In order to find the best value for regularization we applied a regularization function to a sequence of values from 0 to 7.5 with increments of 0.25. The value which returned the smallest RMSE was used for our linear model.

```
regularization <- function(lambda, trainset, testset){

  # Calculating mean

  mu <- mean(trainset$rating)

  # Calculating the movie effect b_i

  b_i <- trainset %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

  # Calculating the user effect b_u

  b_u <- trainset %>%
    left_join(b_i, by="movieId") %>%
    filter(!is.na(b_i)) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

  # Calculating the genre effect b_g

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

  # Calculating the time effect b_t

  b_t <- trainset %>%
    left_join(b_i, by = "movieId")%>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by ="genres") %>%
    group_by(year_of_rating) %>%
    summarize(b_t = mean(rating - mu - b_i - b_u - b_g)/ (n() + lambda))

  # Prediction with mean, b_i, b_u, b_g and b_t

  predicted_ratings <- testset %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_t, by = "year_of_rating") %>%
    filter(!is.na(b_i), !is.na(b_u), !is.na(b_g), !is.na(b_t)) %>%
```

```
    mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
    pull(pred)

  return(RMSE(predicted_ratings, testset$rating))
}
```

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



Comparison of Results
Project Goal: RMSE = 0.8649

Approach	RMSE	MSE	MAE
Regularized Linear Model	0.8638	0.7462	0.6680

In comparison to the unregularized model, the returned RMSE is little better. However the value has only changed in the 3rd decimal place.

Matrix Factorization SGD

As already discussed in the methods section using the library "recosystem" provides the opportunity to set tuning parameters. Different tuning parameters where tested as you can see in the code below:

```
train_data <-  with(train_set, data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))
test_data  <-  with(test_set,  data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))

# Create the model object

ref_object <-  recosystem::Reco()

# Select the best tuning parameters

opts <- ref_object$tune(train_data, opts = list(dim = c(20, 30, 40),
                                        lrate = c(0.05, 0.075, 0.1, 0.15),
                                        costp_l2 = c(0.01, 0.1),
                                        costq_l2 = c(0.01, 0.1),
                                        nthread  = 8,
                                        niter = 10))


# Train the algorithm

ref_object$train(train_data, opts = c(opts$min, nthread = 8, niter = 30))

# Save values for predictions in a vector

pred_reco <-  ref_object$predict(test_data, out_memory())
```

The best result could be achieved with the following parameters:

```
## $dim
## [1] 40
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.01
##
## $costq_l1
## [1] 0
##
## $costq_l2
## [1] 0.1
##
## $lrate
## [1] 0.1
##
## $loss_fun
## [1] 0.794553
```

Comparison of Results

Project Goal: RMSE = 0.8649

Approach	RMSE	MSE	MAE
Matrix Factorization	0.7827	0.6126	0.6019

Compared to our regularized linear model, with matrix factorization we achieved a much better result. The RMSE could get reduced by more than 10% which is a huge improvement.

Final Validation

Regularized Linear Model

Applying our regularized linear model on the edx dataset as a training set and the validation set as a test set yielded the following result:

Comparison of Results

Project Goal: RMSE = 0.8649

Approach	RMSE	MSE	MAE
Regularized Linear Model	0.8644501	0.7472740	0.6685913

Matrix Factorization

Applying the recosystem algorithm to the edx dataset as a training set and the validation set as a test set yielded the following result:

Comparison of Results

Project Goal: RMSE = 0.8649

Approach	RMSE	MSE	MAE
Regularized Linear Model	0.8645	0.7473	0.6686
Matrix Factorization Validation	0.7787	0.6063	0.5989

Conclusion

As a first step we downloaded the 10M Movielens dataset. We executed several preparation and cleaning steps to transfer the dataset in our desired format. We analyzed the dataset and provided some visualization steps in order to find some insights that might help to build a linear model. As a first simple approach we implemented a random model which delivered as expected a very big value of our loss function. As a second approach we started to build a linear model step by step including several effects which improved our linear model with greater and lesser success. It turned out that including a movie bias and a user bias enhanced our model significantly while the addition of a genre effect only showed a small improvement. We also tried to investigate if there is a time effect based on the release date of the movies. Including a time effect in our model didn't change the returned value of our loss function at all. A subsequent regularization step could further improve our model, so that we were able to push down the value for our loss function to 0.8645. We also applied the "recosystem" matrix factorization algorithm which is provided by LIBMF a C++ based library to our datasets. It turned out that we achieve with matrix factorization and stochastic gradient descend much better predictions with a resulting root mean squared error of 0.7787.

Limitations

The dataset 10 M ratings and 70 K users is quite big to execute certain machine learning algorithms. For instance the attempt to build a User x Movie matrix for the whole dataset yielded an out of memory error for my computer. The dataset itself has gives some limitation since we can only work with the information of userID, movieID, genre and release date and the date of review. Modern recommendation systems probably include also other user specific information such as age or sex and other movie specific information like actors, directors etc.

The model only works for existing users, movies and ratings which means that it has to be updated permanently which might be time consuming especially when there is huge movie and user base. Another issue is described by the so called cold start problem which basically means that whenever there is a new user you can't use any explicit information for a recommendation since there is none. The same issue exists for new movies that get added to the movie base, since there is no ratings yet on this movie. In order to overcome these problems it might be useful to include other predictors as long there is no ratings.

Future Ideas

Recommendation systems based on matrix factorization represent the current state of the art. A promising future approach might be a Recurrent Neural Network with the ability to include time as a predictor. Incorporating time into a recommendation system might be important. During Christmas time for instance it is much more likely that people would like to get recommended a Christmas movie. A matrix factorization based algorithm is not able to include time effects like this.

References

1. Rafael A. Irizarry (2019), Introduction to Data Science: Data Analysis and Prediction Algorithms with R

2. Yixuan Qiu (2017), recosystem: recommendation System Using Parallel Matrix Factorization

3. W.-S. Chin, B.-W. Yuan, M.-Y. Yang, Y. Zhuang, Y.-C. Juan, and C.-J. Lin. LIBMF: A Library for Parallel Matrix Factorization in Shared-memory Systems. JMLR, 2016.

4. W.-S. Chin, Y. Zhuang, Y.-C. Juan, and C.-J. Lin. A Fast Parallel Stochastic Gradient Method for Matrix Factorization in Shared Memory Systems. ACM TIST, 2015.

5. https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429

6. https://cran.r-project.org/web/packages/recosystem/index.html

7. https://grouplens.org/datasets/movielens/10m/