Predicting movie ratings in Movielens data

Diana Ojeda-Revah

1 Introduction

This report The data was downloaded from the website http://files.grouplens.org/datasets/movielens/ml-10m.zip. It consists of 6 variables:

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

The objective of the analysis is to predict the variable *rating* (response) from the other 5 predictors.

The variables in the dataset are:

- *userId* identifies the users that gave a rating. It is an integer in the data set, but it is considered a qualitative variable in the analysis. The same value for the variable appears several times, as it is for the same user rating a *different* movie.
- *movieId* is a label for each movie. Most id's appear several times, as many users rated several movies. The type of the variable is numeric but it is a qualitative variable.
- *rating* is a numeric variable between 0 and 5, but with discrete values of steps 0.5, representing the rating the user on that row gave the movie on that same row.
- *title* is the title along with the year the movie was released. Although it is a character variable, the year can be extracted to be used as an additional predictor.
- *timestamp* is a numeric value that contains the data and time when the review was given, and the variable
- *genres* is the genre of the movie, a qualitative variable.

In the following sections, the data is initially explored in order to find patterns to help build a model. The data used to build the model is further partitioned in order to have a data to train and to test different models, without using the validation model. The model is built stepwise, gradually adding complexity. In the last section

2 Exploratory analysis and data engineering.

2.1 Creating new variables.

Upon inspection, we can see that the variable *title* has additional information we can use for our analysis; the year the movie came out. A new variable called *year* is created using the stringr package. It should be done carefully, as the title itself may contain a year. Fortunately the year is at the end of the title, and every title includes the year.

Other two auxiliary variables are created from *timestamp*, one is the year of the review: *yearreview* and the other one is the month of the review: *monthreview*.

2.2 Description of the data and exploratory analysis

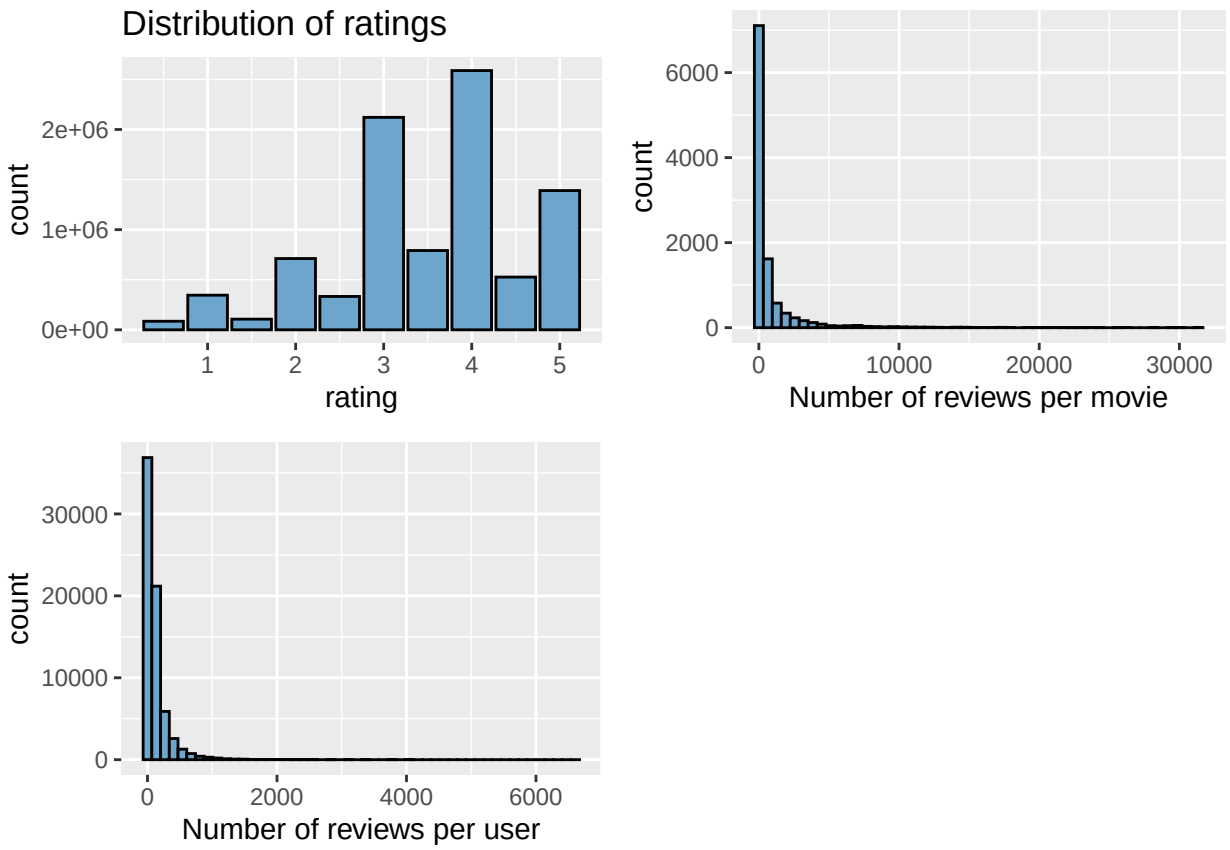
As it was presented in the previous section, the dataset consists of 9,000,055 rows and 6 columns. There are 9,000,055 data ratings corresponding to 10677 distinct movies, 69878 distinct users and 797 genres.

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

```
##   users movies genres
## 1 69878  10677    797
```

```
##   mean_rating sd_rating
## 1    3.512465  1.060331
```

The data contains a list of movies and a list of users. Each user did not rate all the movies present, and neither movie was rated by all of the users. Some users rated many movies, some just a few, the same with the movies. So the first thing we can explore is the distribution of the number of ratings per movie and per user, as shown in the following histograms.



The first plot shows the distribution of ratings. The ratings are not given in a continuous scale, but in steps of 0.5. The plot shows that most users gave the rating in whole numbers, and the mode is a rating of 4. The number of reviews per movies and and per user show that there is a variation in the number of reviews. This is further understood by exploring the summary statistics of the number of reviews per user and per movie.

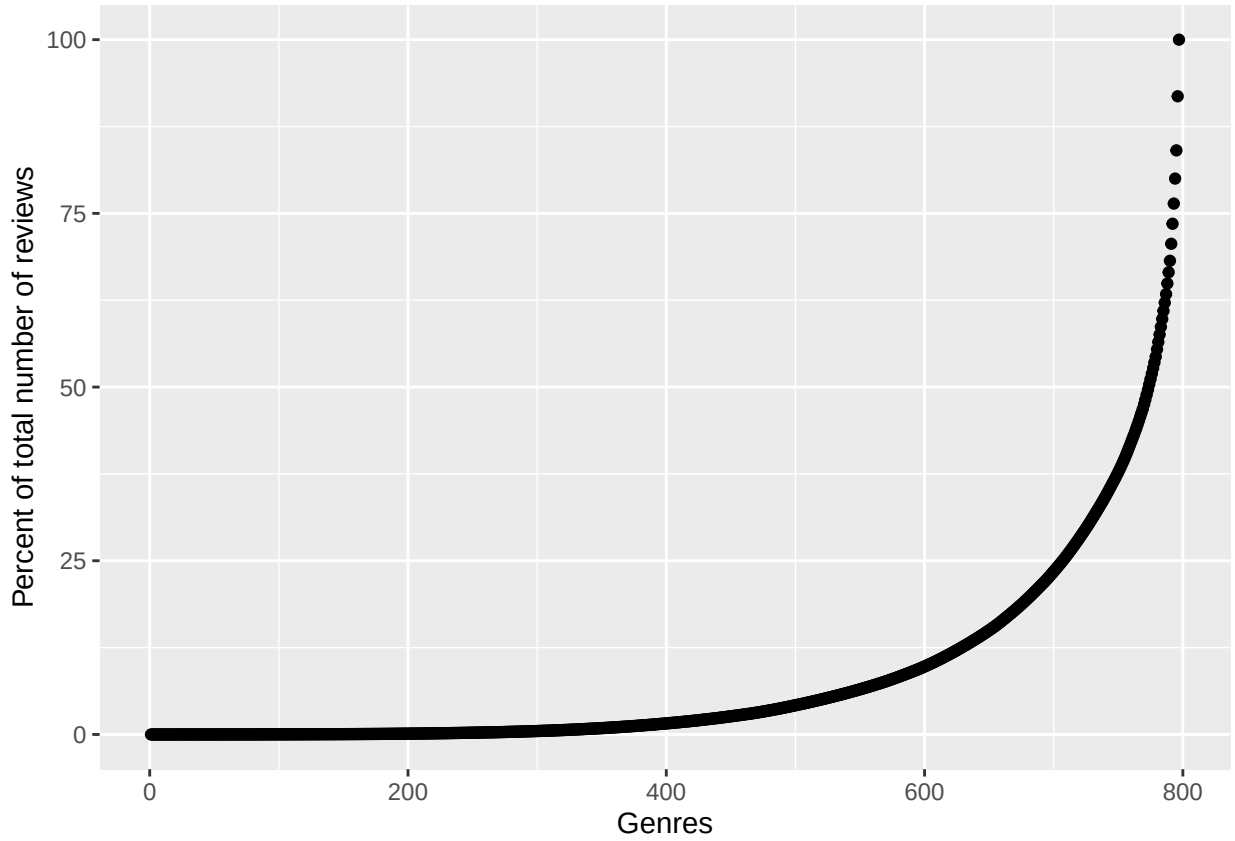
```
## # A tibble: 2 x 6
##   Variable   Min   Max First_quantile Median Third_quantile
##   <chr>    <int> <int>          <dbl>  <dbl>          <dbl>
## 1 movieId      1 31362             30    122            565
## 2 userId      10  6616             32     62            141
```

The movie (or movies) that has the least ratings got one and the maximum number of ratings a movie got was 31,362. 75% of the movies got at least 30 ratings. The users in the database rated at least 10 movies, and the maximum number of ratings for a user was 6,616.

The dataset has a variable describing the genres of the movie. There are 797 different values for the variable genres. Most of them are combinations of two or more "classic" genres (for example "Action|Crime|Thriller").

```
##   n_distinct(genres)
## 1                797
```

In the following plot we graph the cummulative percentage of reviews per genre.



From this plot we can see that most of the reviews are concentrated in a very few genres. Half of the total number of reviews are concentrated in only 24 genres.

```
## # A tibble: 1 x 1
##       s
##   <int>
## 1    24
```

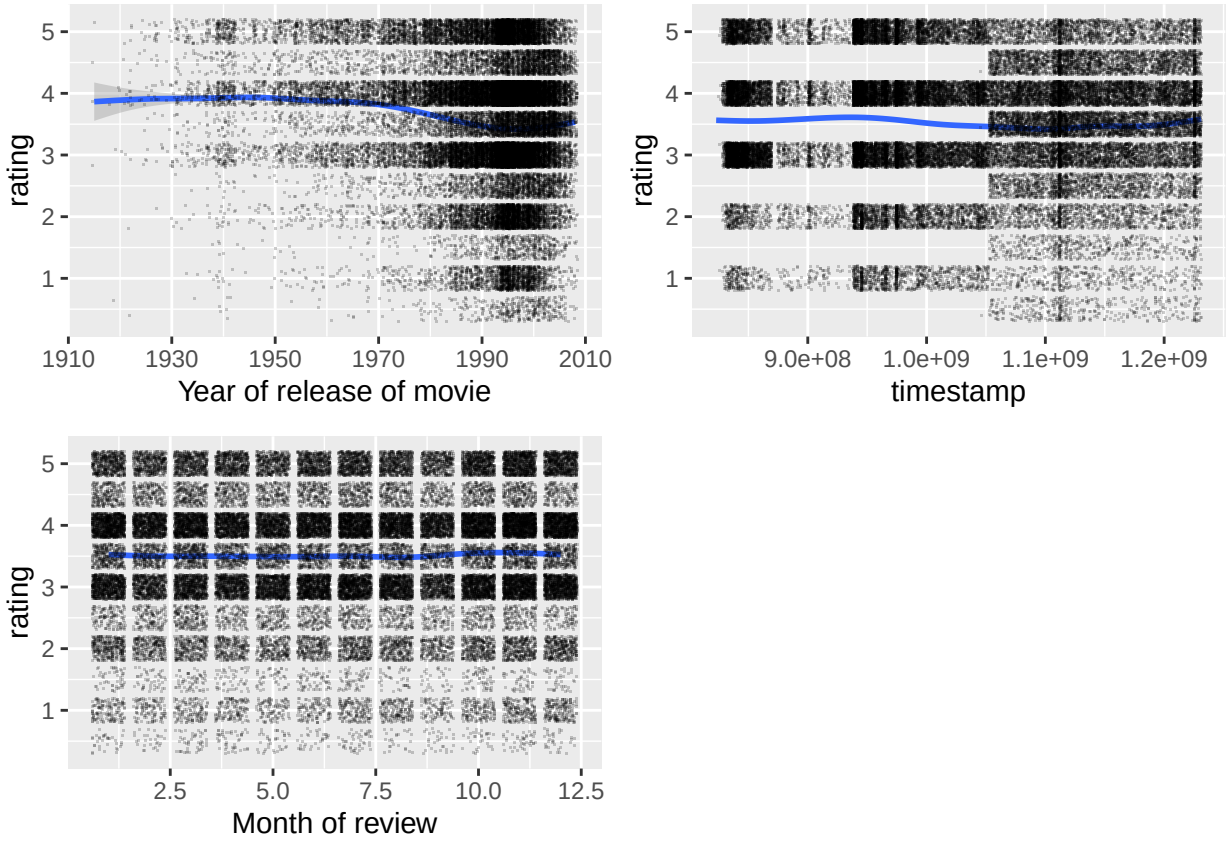
It is important to study these distribution of ratings to help us better model the behavior. We may find the case that users, movies or genres with very few ratings could be having too much influence in the predictions and introducing an error.

The last aspect we are going to explore are time variations. There are two time variables in the dataset. The timestamp that is included in the original dataset, and the one that was obtained from the titles. The timestamp indicates the date and time when the review was given and the new variable "year" indicates the year the movie was released.

A random sample of 100,000 is used for creating a scatterplot of year (year the movie was released) vs rating,

along with a smoothing.

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



There seems to be an effect on the ratings from the time the movie was released, apparently new movies are, on average, rated a bit lower. But there seems to be no effect on the variable timestamp (the date and time when the rating was made) nor a seasonal effect (month of the review).

3 Model building.

The model was built step by step in a matter similar to the way it was worked on the course, following these steps:

- Data was split into test and train variables. In the rest of the document, we will refer to these as *train* and *test* sets. The set that was initially kept apart, the one for final RMSE, will be referred to as validation set.
- A first model with only the global mean is built, in order to establish a baseline for the rmse to improve upon.
- A second model is built adding movie effects.
- A third model is built, adding movie and user effects.
- A fourth model is built, adding regularization and genre effects.
- A fifth and final model is built, adding the effect of year the movie was released, using spline smoothing.

The models are evaluated with a residual mean squared error (RMSE). The rating by the user u for the

movie for the movie i is given by $y_{u,i}$ and the predictions are given by $\hat{y}_{u,i}$. The definition of RMSE is:

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

And we define it n R as:

The models that are progressively built, will be evaluated with the test set, and only in the end, the final model will be fit with the whole dataset and evaluated with the validation set. We define the train and test set here:

```
#####################################
# Create partition of edx  set to
# create a train set and a test set
#####################################
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
edxtrain <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in validation set are also in edx set
edxtest <- temp %>%
    semi_join(edxtrain, by = "movieId") %>%
    semi_join(edxtrain, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, edxtest)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres", "year", "yearreview",
```

```
edxtrain <- rbind(edxtrain, removed)

rm(temp)
```

3.1 First Model. Global mean.

To establish a baseline to improve upon, our first model just assumes that all the ratings are equal to the global mean, that is, $Y_{u,i} = \mu + \epsilon_{u,i}$. The model is implemented in R in the following way:

```
# Find global average
# this will be our baseline
mu_hat <- edxtrain %>%
    summarize(m=mean(rating)) %>%
    .$m

#rmse
model1_rmse <- RMSE(mu_hat, edxtest$rating)
model1_rmse
```

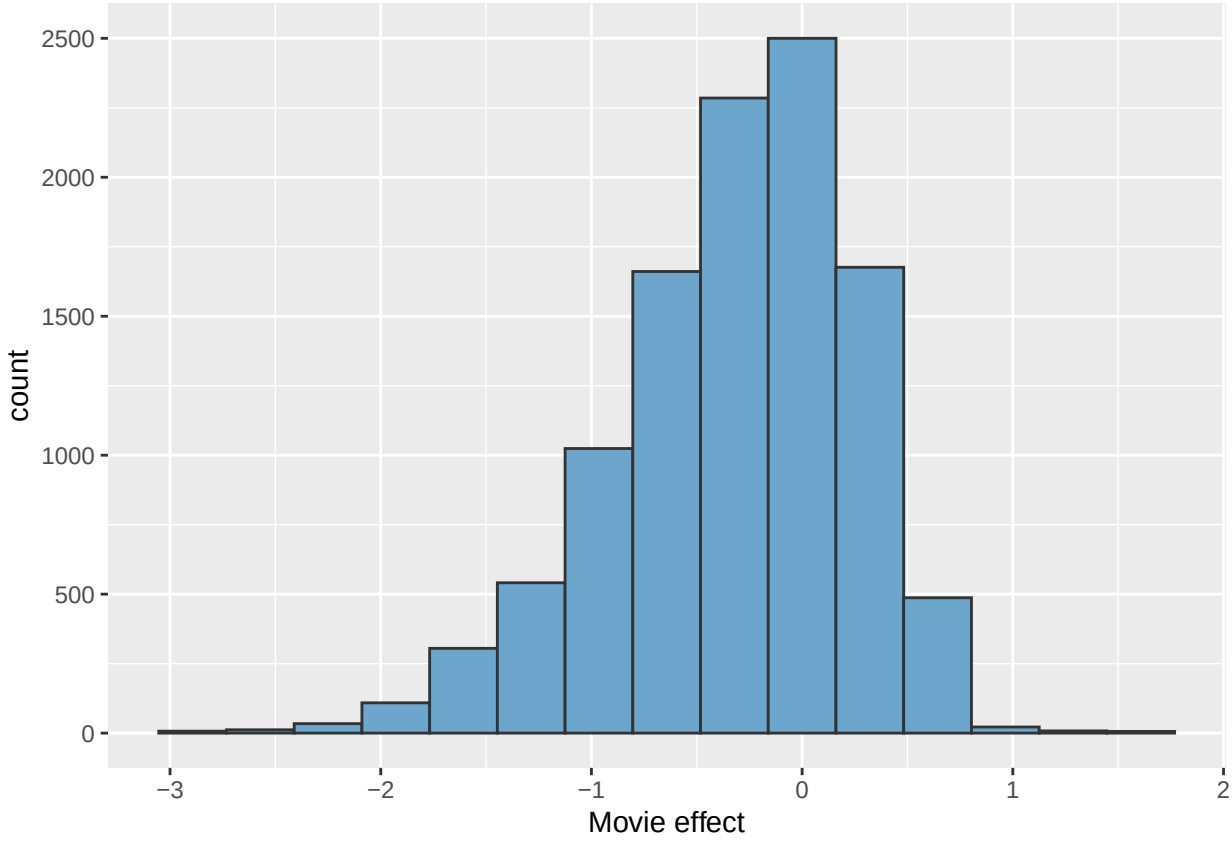
```
## [1] 1.060054
```

So our most basic model yields rmse of `model1_rmse` in the test set.

3.2 Second model. Add movie effects.

We start making our model more complex by examining the variation of the average rating of each movie, after the global mean has been subtracted.

```
# average ratings by movie, after taking general mean
movie_effect <- edxtrain %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu_hat))

# Plot of distribution of  of movie effects
movie_effect %>% ggplot +
    aes(b_i) +
    geom_histogram(bins=15, color=mycolor, fill=myfill) +
    xlab("Movie effect")
```



On the histogram we can see that there is a great variation in the means of the ratings. This leads us to add to the basic model a movie effect, so our new model is given by 1:

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i} \tag{1}$$

We estimate the movie effects, $\hat{b}_i$ by averages that we used to construct the histogram. The estimates of the ratings in the test set are given by sum of the mean in the training set plus the average mean of the corresponding movie, also in the train set. So we get:

```
#Predict on validation set
```

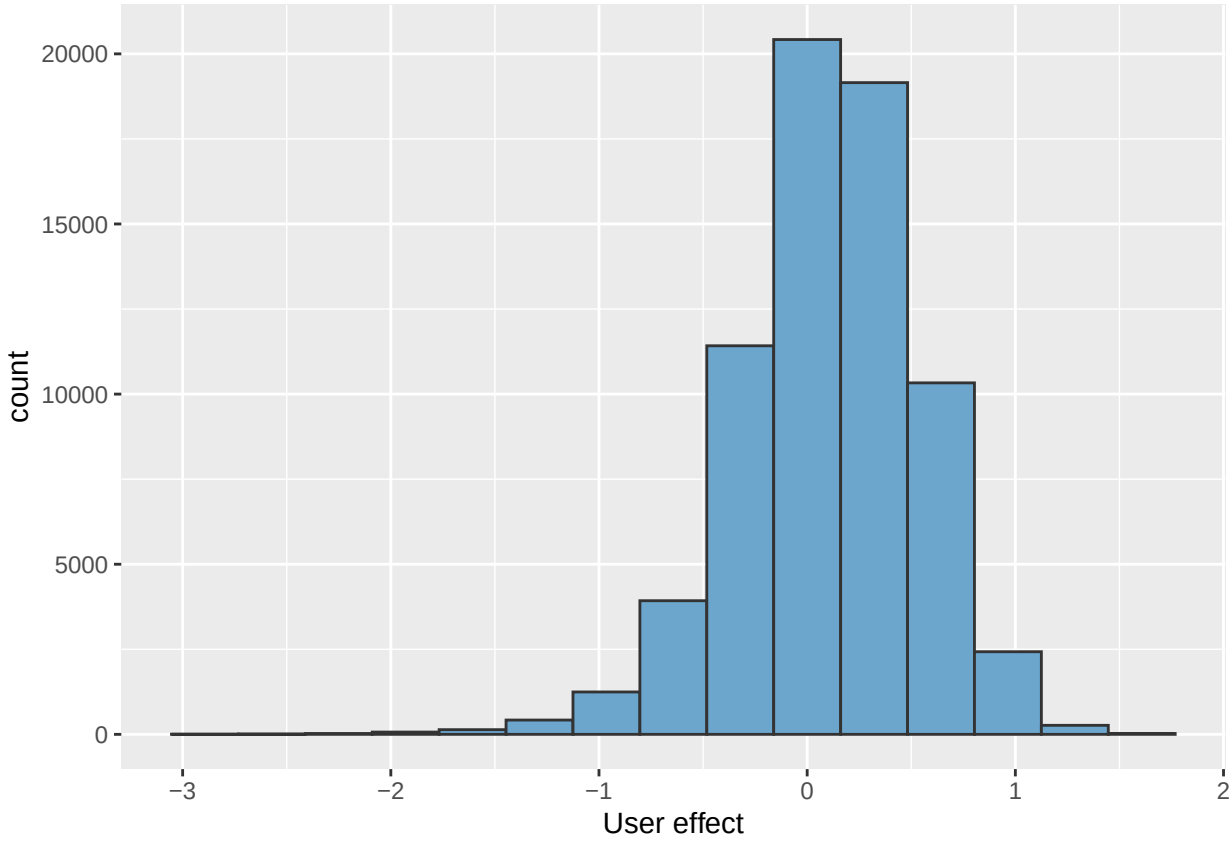
```r
yhat <- left_join(edxtest, movie_effect, by="movieId") %>%
    mutate(yhat=b_i + mu_hat) %>%
    pull(yhat)

# Rmse of model 2.
model2_rmse <- RMSE(yhat ,edxtest$rating)
model2_rmse
```

```
## [1] 0.9429615
```

Our new rmse shows an improvemente over the previous one, to `R model2_rmse`, where we used only the mean.

3.3 Third model. Add user effects.

Exploring the data further, we find that the mean rating of the users is not constant, but follows a distribution in which some users give, on average higher ratings than others, as shown in the following histogram:



In a similar way as the previous model, we add the mean rating per user to the model so our new model is 2:

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i} \tag{2}$$

The user effects are estimated as the average rating per user, after the global mean and movie effects have been substracted:

```
user_effect <- edxtrain %>%
    left_join(movie_effect, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu_hat - b_i))

yhat <- edxtest %>%
    left_join(movie_effect, by="movieId") %>%
    left_join(user_effect, by="userId") %>%
    mutate(prediction = mu_hat + b_i + b_u) %>%
    pull(prediction)

model3_rmse <- RMSE(yhat, edxtest$rating)
model3_rmse
```

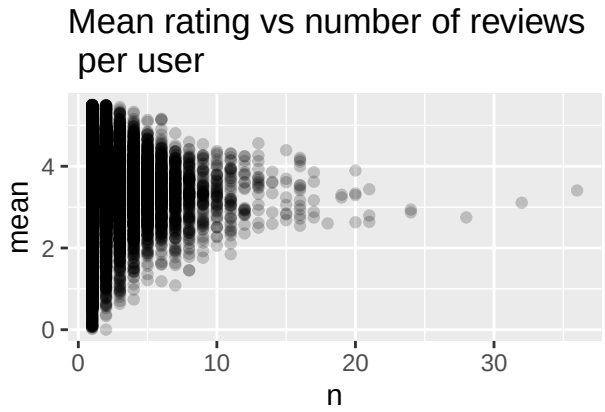
```
## [1] 0.8646843
```

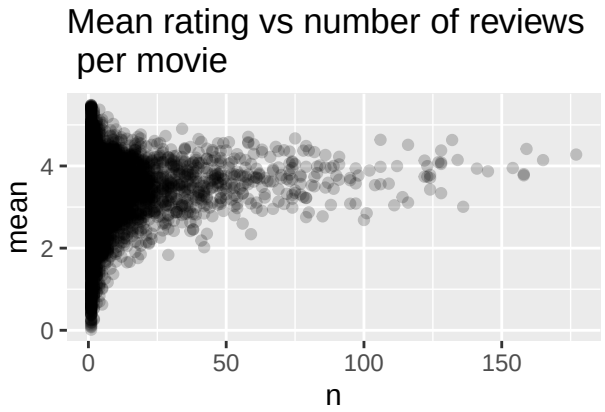
The rmse is further decreased to 0.8646843.

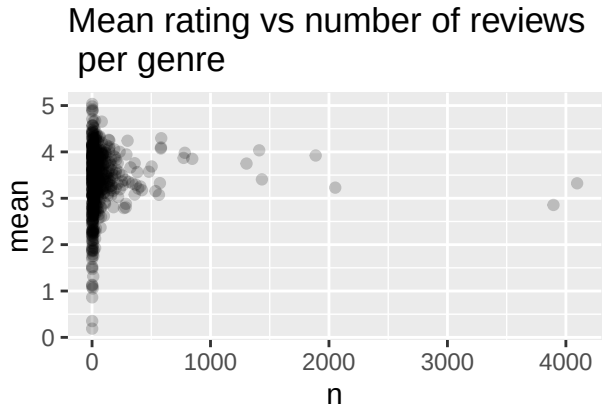
3.4 Fourth model. Add regularization and genre effects.

To improve the model further, we analyze the average rating per user, movie and genre. From the plots we can conclude that the users, movies and genres with the least number of ratings have the most extreme values, and also that the variance of the mean ratings gets larger as the number of users or movies rated gets smaller. There is a way to deal with this problem by using regularization, which introduces a penalty to estimates of the effects, when the number of ratings is small..

```
p4_1 <-edxtrain %>%
    sample_n(50000) %>%
    group_by(userId) %>%
    summarize(mean = mean(rating), n=n()) %>%
    ggplot(aes(n,mean)) +
    geom_jitter(alpha=0.2, width=0, height=0.5) +
    ggtitle("Mean rating vs number of reviews \n per user")

p4_2<-edxtrain %>%
    sample_n(50000) %>%
    group_by(movieId) %>%
    summarize(mean = mean(rating), n=n()) %>%
    ggplot(aes(n,mean)) +
    geom_jitter(alpha=0.2, width=0, height=0.5)+
    ggtitle("Mean rating vs number of reviews \n per movie")

p4_3 <-edxtrain %>%
    sample_n(50000) %>%
    group_by(genres) %>%
    summarize(mean = mean(rating), n=n()) %>%
    ggplot(aes(n,mean)) +
    geom_jitter(alpha=0.2, width=0, height=0.5) +
    ggtitle("Mean rating vs number of reviews \n per genre")

grid.arrange(p4_1, p4_2, p4_3, ncol=2)
```







The model is implemented in the following way: first the lambda that minimizes the rmse on the test set is obtained and then the model is fit in the rest of the data.

```
lambdas <- seq(0, 10, 0.25) #lammbdas to test
rmses <- vector(mode="numeric", length=length(lambdas))
rmses <- sapply(lambdas, function(l){
    mu <- mean(edxtrain$rating)
    b_i <- edxtrain %>%
        group_by(movieId) %>%
        summarize(b_i = sum(rating - mu)/(n()+l))
    b_u <- edxtrain %>%
        left_join(b_i, by="movieId") %>%
        group_by(userId) %>%
        summarize(b_u = sum(rating - b_i - mu)/(n()+l))
    b_g <- edxtrain %>%
        left_join(b_i, by="movieId") %>%
        left_join(b_u, by="userId") %>%
        group_by(genres) %>%
        summarize(b_g =sum(rating -mu - b_i -b_u)/(n()+l))
    predicted_ratings <-
        edxtest %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_g, by="genres") %>%
        mutate(pred = mu + b_i + b_u + b_g) %>%
        pull(pred)
    return(RMSE(predicted_ratings, edxtest$rating))
```

```
})

# plot lambdas against the rmses obtained with the test set.
# Set the lambda to the one that minimizes the error
plot(lambdas, rmses)
l =lambdas[which.min(rmses)]
```

We find that a value of $\lambda = 5$ that minimizes the rmse.

```
# Fit regularized model with lambda obtained, including genre effects
l <- 5
mu <- mean(edxtrain$rating)
b_i <- edxtrain %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
b_u <- edxtrain %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
b_g <- edxtrain %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g =sum(rating -mu - b_i -b_u)/(n()+l))


# Find predictions on test set
predicted_ratings <-
    edxtest %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by="genres") %>%
    mutate(prediction = mu + b_i + b_u +b_g) %>%
    pull(prediction)


# rmse on test set
model4rmse =RMSE(predicted_ratings, edxtest$rating)
model4rmse
```

```
## [1] 0.8638148
```

The rmse is further reduced to a value of 0.8638148.

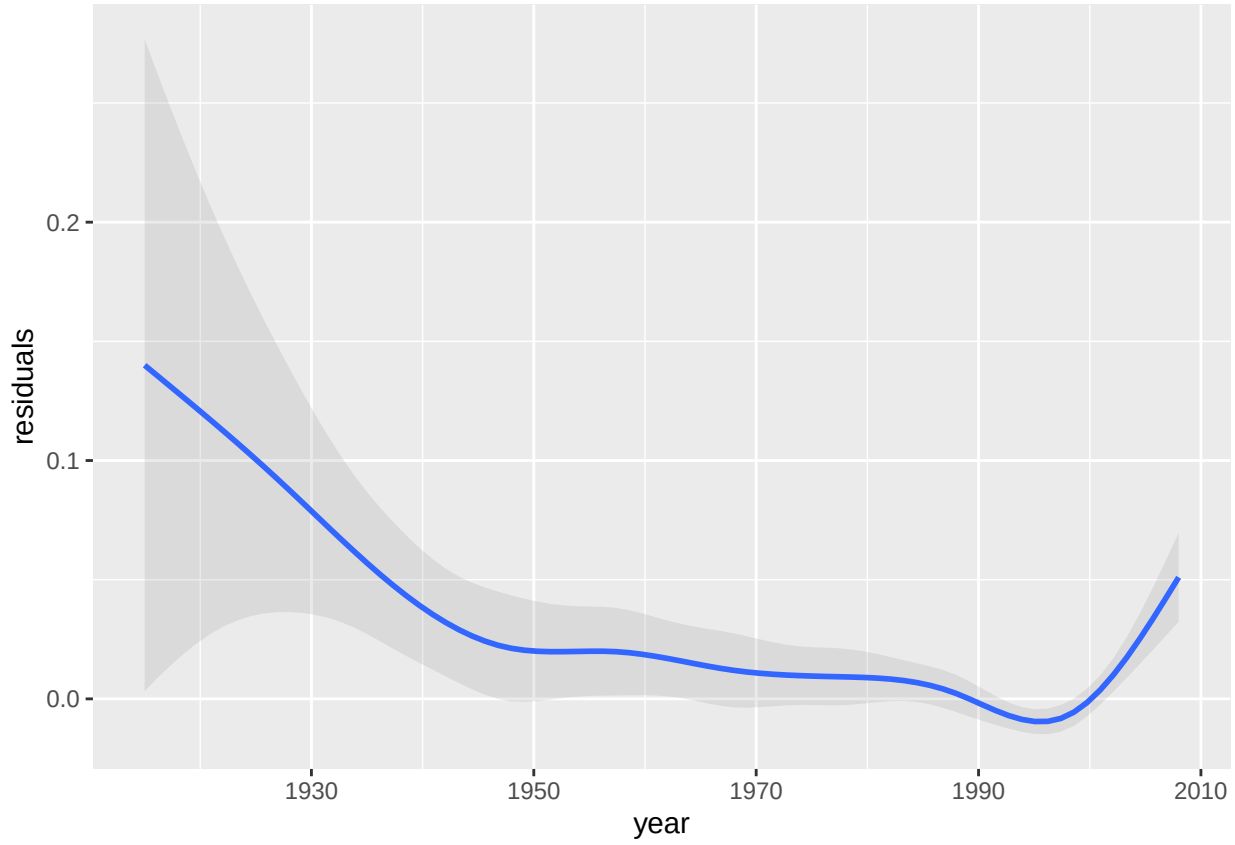
3.5 Fifth model. Add time effects.

In the exploratory section, a smoothed plot of ratings vs time of release showed a probable time effect, with older movies being, on average, rated a bit higher. Now we create a similar plot, but with the residuals from the previous models instead of the ratings:

```
predictions <- edxtrain %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    mutate(prediction = mu + b_i + b_u +b_g) %>%
```

```
    pull(prediction)

#find redisuals on train set
edxtrain <- edxtrain %>% mutate(residuals = rating - predictions)

#plot residuales against year the movie came
set.seed(1)
edxtrain %>% sample_n(200000) %>%
    ggplot() +
    aes(year, residuals) +
    geom_smooth(alpha=.2)
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



For the plot, only a fraction of the total data is used for the smoothing. It is enough to give us an idea that there seems to be a time effect, and highly non-linear.

To model this effect, a penalized cubic splines is selected. A cubic splines model is used instead of a smoothing spline or loess for computational reasons, given that it is the one that takes the shortes time. Penalties are used given that older movies, though on average are better rated, have much fewer ratings, therefore higher variance. The model is also fit to the timestamp variable.

For this model, the package *mgcv* is used with caret. The tuning parameter for this model is the number of knots, k. (referencia). Several knots are tested for training the data. The tuning is done in two parts, first with a broader grid and then, a second loop is run with a finer grid. As this process takes approximately an hour to run, the code for obtaining the optimal k was not run during the knitting of this document, although the code is included in the Rmd file and in the R file.

The splines model is applied to the **residuals** that were generated after fitting model 4, and then added to the fit. So model 4 has to also be fitted to the test model and residuals generated.

```
#Two loops are run in order to find the k parameter in the
#spline model that mimimizes the error on the test set.

#values of k to test
ks <- expand.grid(k1=c( 12,14, 16,18), k2=c(8, 10, 12))

errors_spline1 <- vector(mode="numeric", length=nrow(ks))

#loop that tests values of k
for (i in (1:nrow(ks))){
    print(ks[i,1])
    print(ks[i,2])
    fit <- gam(residuals ~ s(year, bs="cr", k=ks[i,1]) +
                   s(timestamp, bs="cr", k=ks[i,2]),
                   data=edxtrain)

    ehat <-predict(fit, edxtest)

    yhat <- edxtest %>%
            left_join(b_i, by="movieId") %>%
            left_join(b_u, by="userId") %>%
            left_join(b_g, by="genres") %>%
            mutate(prediction = mu + b_i + b_u +b_g) %>%
            mutate(prediction2 = prediction + ehat) %>%
            pull(prediction2)

    modeltime <- RMSE(yhat, edxtest$rating)
    errors_spline1[i] <-modeltime
}


ks2 <- expand.grid(k1=c(15,16, 17), k2 = c(9,10,11))
errors_spline2 <- vector(mode="numeric", length=nrow(ks))

#loop that tests values of k
for (i in (1:nrow(ks))){
    print(ks[i,1])
    print(ks[i,2])
    fit <- gam(residuals ~ s(year, bs="cr", k=ks[i,1]) + s(timestamp, bs="cr", k=ks[i,2]),
               data=edxtrain)

    ehat <-predict(fit, edxtest)


    yhat <- edxtest %>%
        left_join(b_i, by="movieId") %>%
        left_join(b_u, by="userId") %>%
        left_join(b_g, by="genres") %>%
        mutate(prediction = mu + b_i + b_u +b_g) %>%
        mutate(prediction2 = prediction + ehat) %>%
        pull(prediction2)
```

```
    modeltime <- RMSE(yhat, edxtest$rating)
    errors_spline2[i] <-modeltime
}
```

The model training return values of $k = 15$ for year of release and $k = 11$ for timestamp, which are fed to the final model:

```
#Fit a final model for the RESIDUALS with the correct values for K
spline_time <- gam(residuals ~ s(year, bs="cs", k=15) + s(timestamp, bs="cs", k=11),
           data=edxtrain)

# fit model on the test set
ehat <-predict(spline_time, edxtest)

#find predicions of test set
yhat <- edxtest %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    mutate(prediction = mu + b_i + b_u +b_g) %>%
    mutate(prediction2 = prediction + ehat) %>%
    pull(prediction2)

# check rmse for new model in TEST set
model5rmse <- RMSE(yhat, edxtest$rating)
model5rmse
```

```
## [1] 0.8635193
```

The rmse, evaluated on the test set, is further reduced. This is the complete model that will be fit in the complete edx set and evaluated in the validation set, on next section.

4 Getting final RMSE with validation set.

On the previous sections, a model was built step by step. The part of the data that was labelled "edx" was divided in a training and testing set. All the training was done in the training data, labelled *trainset* and the evaluations of the RMSE on the test set. The training resulted in a model made of the sum of the mean, a user bias, a movie bias and a gender bias, all minimized with a penatly parameter, plus a penalized cubic splines smoothing of the effect of year of release.

This model is finally fit to the edx data, taking the exact steps we took when building it, and using the values of λ, the penalty paramenter and k, the tuning parameter obtained in the previous sections. So our model is:

```
# Fit regularized model with lambda obtained, including genre effects
l <- 5
mu_full <- mean(edx$rating)
b_i_full <- edx %>%
    group_by(movieId) %>%
    summarize(b_i_full = sum(rating - mu_full)/(n()+l))
b_u_full <- edx %>%
    left_join(b_i_full, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_full = sum(rating - b_i_full - mu_full)/(n()+l))
b_g_full <- edx %>%
```

```
    left_join(b_i_full, by="movieId") %>%
    left_join(b_u_full, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g_full =sum(rating -mu_full - b_i_full -b_u_full)/(n()+1))




#Find predictions to find residuals
predictions <- edx %>%
    left_join(b_i_full, by="movieId") %>%
    left_join(b_u_full, by="userId") %>%
    left_join(b_g_full, by="genres") %>%
    mutate(prediction = mu_full + b_i_full + b_u_full +b_g_full) %>%
    pull(prediction)

edx <- edx %>% mutate(residuals = rating - predictions)


#Predictions and residuals on validation set for splines
predictionst <- validation %>%
    left_join(b_i_full, by="movieId") %>%
    left_join(b_u_full, by="userId") %>%
    left_join(b_g_full, by="genres") %>%
    mutate(prediction = mu_full + b_i_full + b_u_full +b_g_full) %>%
    pull(prediction)

#find redisuals on train set
validation <- validation %>% mutate(residuals = rating - predictionst)

#Fit splines on time data on edx set (train set.)

fit <- gam(residuals ~ s(year, bs="cs", k=15) + s(timestamp, bs="cs", k=11),
           data=edx)

ehat <-predict(fit, validation)


yhat <- validation %>%
    left_join(b_i_full, by="movieId") %>%
    left_join(b_u_full, by="userId") %>%
    left_join(b_g_full, by="genres") %>%
    mutate(prediction = mu_full + b_i_full + b_u_full +b_g_full) %>%
    mutate(prediction2 = prediction + ehat) %>%
    pull(prediction2)

finalrmse <- RMSE(yhat, validation$rating)
finalrmse
```

```
## [1] 0.8641569
```

So our final RMSE is 0.8641569.

5 Conclusions.

Our final model yields a rmse of 0.8641569, which is within the requirements. There is considerable variation in the means of the ratings within the users. Some users tend to rate, on average, higher than others. Movies with few ratings tend to be rated much higher or lower than the average.

Although the original data only had 4 predictor, we were able to extract from the titles further information that turned out to be useful for our analysis. The model presented in this report yields an rmse that is within the goals of the analysis. Yet, there are some further improvements that could be attempted to improve upon the model we presented.

The variable *genres* has 797 different factors, some with very few ratings. The values are composed of one or more major genres, such as "drama|romance", or "children|animation|sci-fi". Being able to group some of these genres in order to reduce their number could potentially permit us to better exploit this variable. The variable could either be reduced to one major genre or divided into two variables. Having a reduced number of factors for genre could potentially permit us to find interactions with users and factor, interactions of genres with time and even build a matrix to perform singulare value decomposition. However, the difficulty lies in the fact that the genres on each value of the factor are ordered alphabetically, not in order of importance, making it impossible to choose one or two major genres that represent the movie's principal genre.

Another way to address this is to build several binary variables that indicate the presence or absence of certain elements in the movie, such as fantasy, comedy or romance and use user preference of those elements to better predict ratings.