Movie Recommendation System

cgoldner

2021-07-04

Contents

1 Overview

1.1 MovieLens 10M dataset

The MovieLens 10M dataset includes 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. It was released in January 2009 and can be found under

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

Each movie can be rated from 0.5 up to 5 stars in steps of size 0.5.

1.2 Goal of the project

Our goal is to build a data-driven model which predicts user ratings of movies based on the data provided by the MovieLens 10M dataset. We refer to this model as *Movie Recommendation System*. Throughout the project, we fix the following notation: Let $Y_{u,i}$ denote the rating of user u of movie i. So our goal is to calculate $\hat{Y}_{u,i}$, an estimate of $Y_{u,i}$ for each user u and each movie i.

1.3 Key steps

The most important steps performed in order to build the Movie Recommendation System are:

(1) Download the data, convert it into a data frame and split off a test set that is only used for evaluation purposes.

(2) Analyze the data to detect effects/pattern useful for prediction.

(3) Combine the observed effects into a mathematical model.

(4) Estimate the different parameters of the model.

(5) Evaluate the model on the test set.

2 Methods

2.1 Loss function

To evaluate the Movie Recommendation system, the root mean squared error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

is used, where N denotes the number of observations, y_i denotes the rating for movie i and $\hat{y}_i$ denotes our predicted rating for movie i. Notice that the RMSE is in the same units as the ratings of the movies. Thus an RMSE of e.g. 0.7 means that, on average, predictions of user ratings may deviate ± 0.7 stars from the true user's rating.

2.2 Data wrangling

The zipped data we use was downloaded from

http://files.grouplens.org/datasets/movielens/ml-10m.zip.

We transform it into a data frame with meaningful column names:

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

Each row represents one observation which is a rating of a movie by a user. Each user has a unique userId. Moreover, each movie can be rated by a different number of users and each user can rate a different number of movies.

In order to test and train our model appropriately, a test set called `validation` which consists of 10% of our data is created. The remaining data is used for training and is called `edx`. Its first entries look like the following:

```
## # A tibble: 3 x 7
##   userId movieId rating timestamp title     genres                         year
##    <int>   <dbl>  <dbl>     <int> <chr>     <chr>                         <int>
## 1      1     122      5 838985046 Boomerang Comedy|Romance                 1992
## 2      1     185      5 838983525 Net, The  Action|Crime|Thriller          1995
## 3      1     292      5 838983421 Outbreak  Action|Drama|Sci-Fi|Thriller  1995
```

2.3 Data analysis

In the following, an analysis of the data is provided. Our goal is to detect patterns/effects present in the data, which a final model of the data needs to address.

2.3.1 Movie effect

The estimated average rating of all users on all movies is

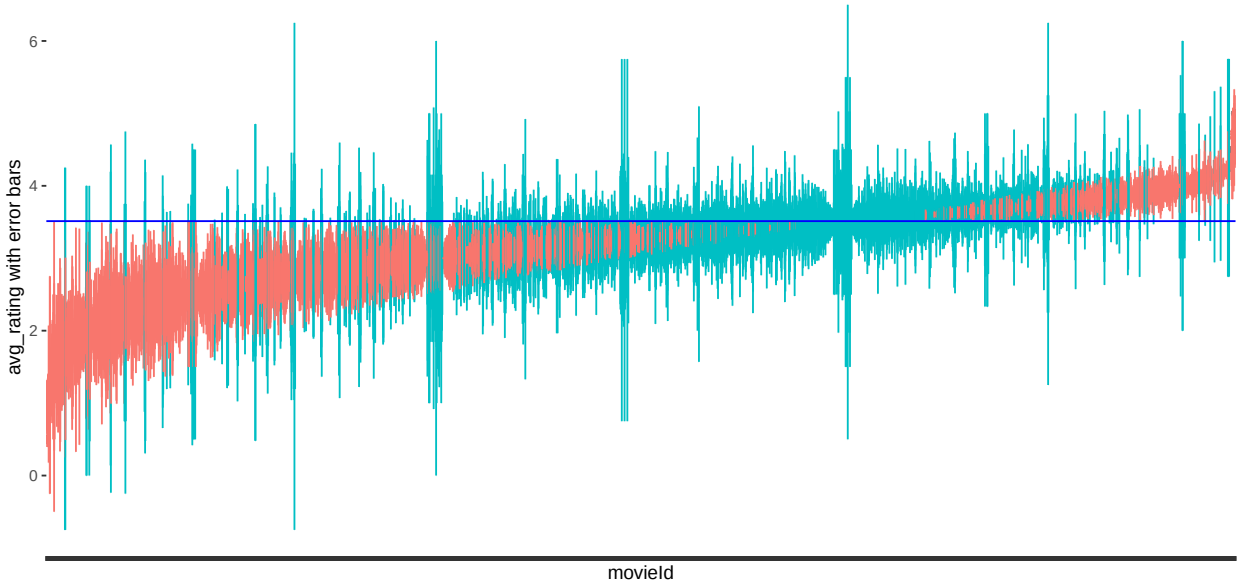
$$\hat{\mu} = 3.512465.$$

The following plot shows the average rating of *each movie* and its error bars. The blue line indicates the estimated overall average rating $\hat{\mu}$. The plot suggests that there is a movie-specific effect that influences a movie's rating since many error bars (the red ones) do not contain $\hat{\mu}$. We refer to this effect as *Movie effect*.

```
movie_infos <- edx %>% group_by(movieId) %>%
  mutate(total_number_of_ratings = n(),
         avg_rating = mean(rating),
         se = sd(rating)/sqrt(n())) %>%
  slice(1) %>%
  select(-rating, -userId, -timestamp)

mu_hat <- mean(edx$rating)

movie_infos %>%
  mutate(ymin = avg_rating - 2*se,
         ymax = avg_rating + 2*se) %>%
  mutate(included = ifelse(mu_hat >= ymin & mu_hat <= ymax, TRUE, FALSE)) %>%
  ggplot(aes(x = reorder(movieId, avg_rating, FUN = mean),
             y = avg_rating, group = movieId,
             ymin = avg_rating - 2*se,
             ymax = avg_rating + 2*se,
             color = included)) +
  geom_point(show.legend = FALSE) +
  geom_errorbar(show.legend = FALSE) +
  scale_x_discrete(labels = NULL) +
  labs(x = "movieId", y = "avg_rating with error bars") +
  geom_hline(yintercept = mu_hat, color = "blue")
```



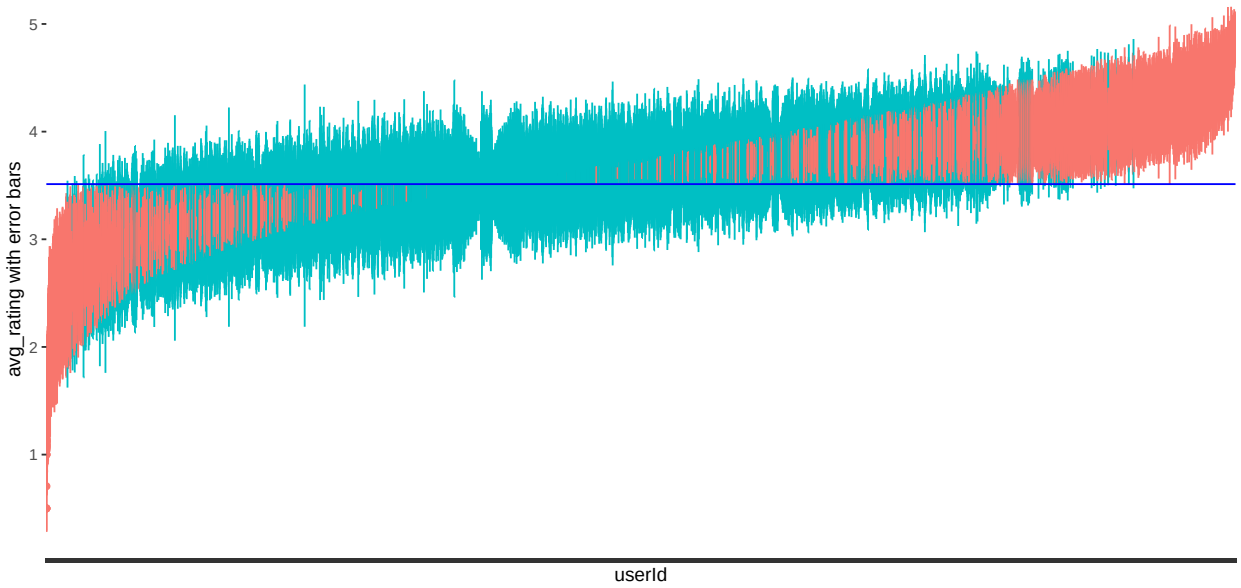
To capture this effect, we introduce a *movie effect bias* b_i. Notice that the plot above shows that most movies are, on average, rated below average.

2.3.2 User effect

In the same way the Movie effect was discovered, a user-specific effect, called *User effect* can be detected. For that, see the following plot, where the blue line indicates $\hat{\mu}$.

```
user_infos <- edx %>% group_by(userId) %>%
  mutate(total_number_of_ratings = n(),
         avg_rating = mean(rating),
         se = sd(rating)/sqrt(n())) %>%
  slice(1) %>%
  select(-rating, -movieId, -timestamp, - year, -genres, -title)

user_infos %>%
  mutate(ymin = avg_rating - 2*se,
         ymax = avg_rating + 2*se) %>%
  mutate(included = ifelse(mu_hat >= ymin & mu_hat <= ymax, TRUE, FALSE)) %>%
  ggplot(aes(x = reorder(userId, avg_rating, FUN = mean),
             y = avg_rating,
             group = userId,
             ymin = avg_rating - 2*se,
             ymax = avg_rating + 2*se,
             color = included)) +
  geom_point(show.legend = FALSE) +
  geom_errorbar(show.legend = FALSE) +
  scale_x_discrete(labels = NULL) +
  labs(x = "userId", y = "avg_rating with error bars") +
  geom_hline(yintercept = mu_hat, color = "blue")
```

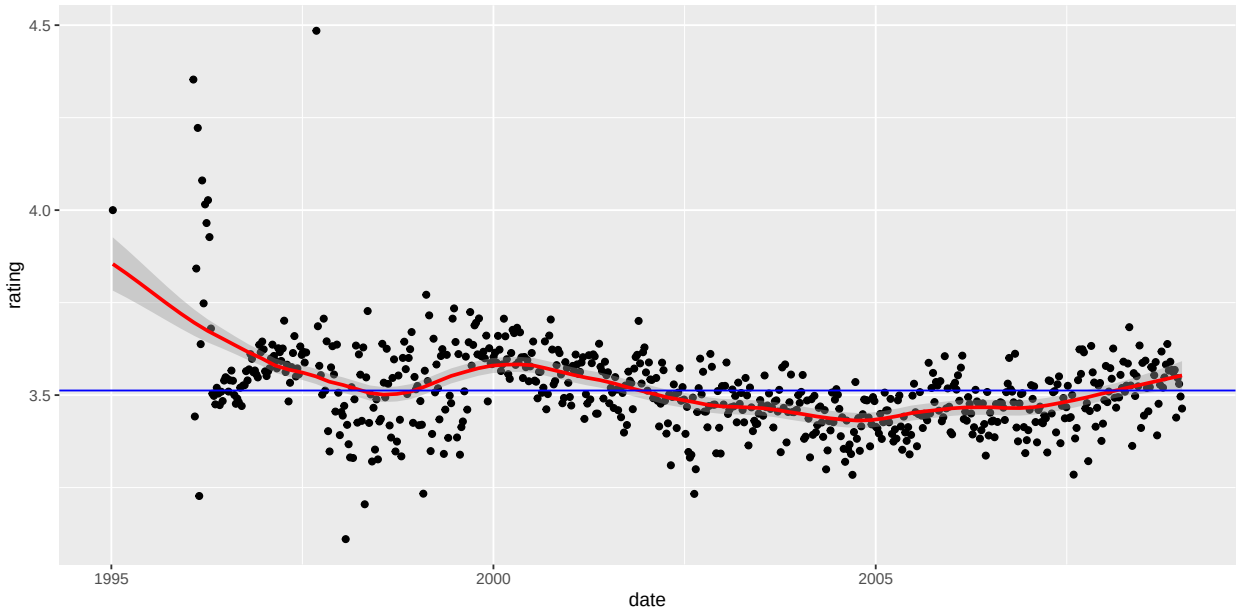


Denote the *user effect bias* by b_u. Notice that most users, on average, rate above average.

2.3.3 Time effect on average rating

Notice that the average rating actually depends on the time as the following plot illustrates. We call this effect the *Time effect on average rating*.

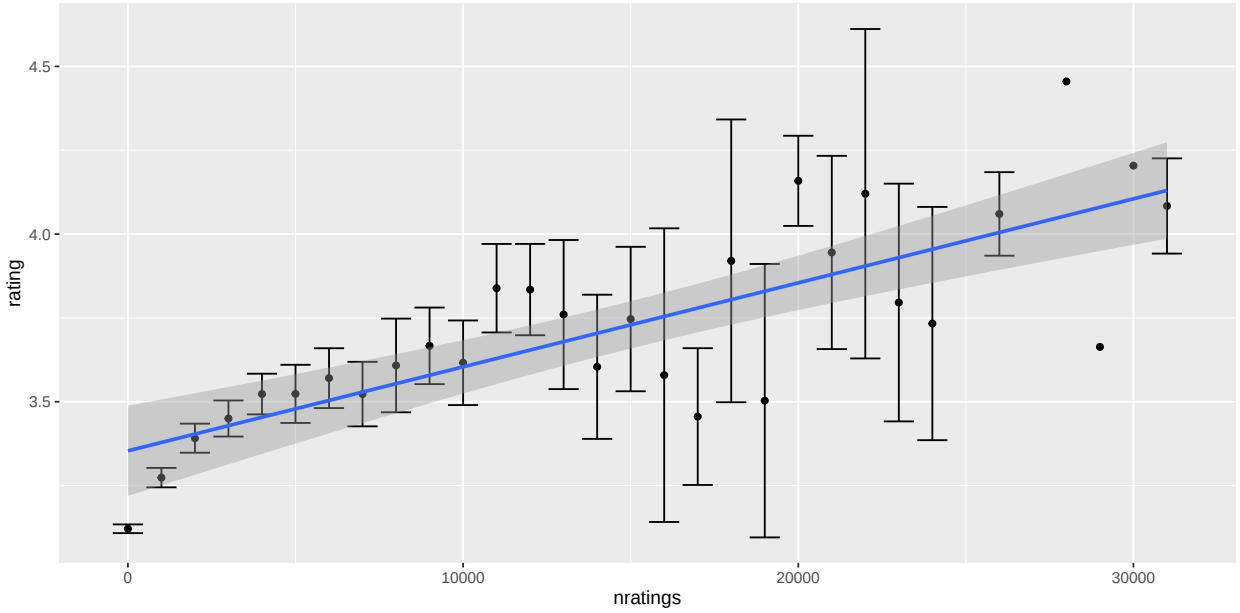
```
edx %>% mutate(date = as_datetime(timestamp)) %>%
  mutate(date = round_date(date, unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth(method = "loess", span = 0.2, method.args = list(degree=1), color = "red") +
  geom_hline(yintercept = mu_hat, color = "blue")
```



2.3.4 Popularity effect

Observe the positive correlation between the number of ratings `nratings` and ratings of movies:

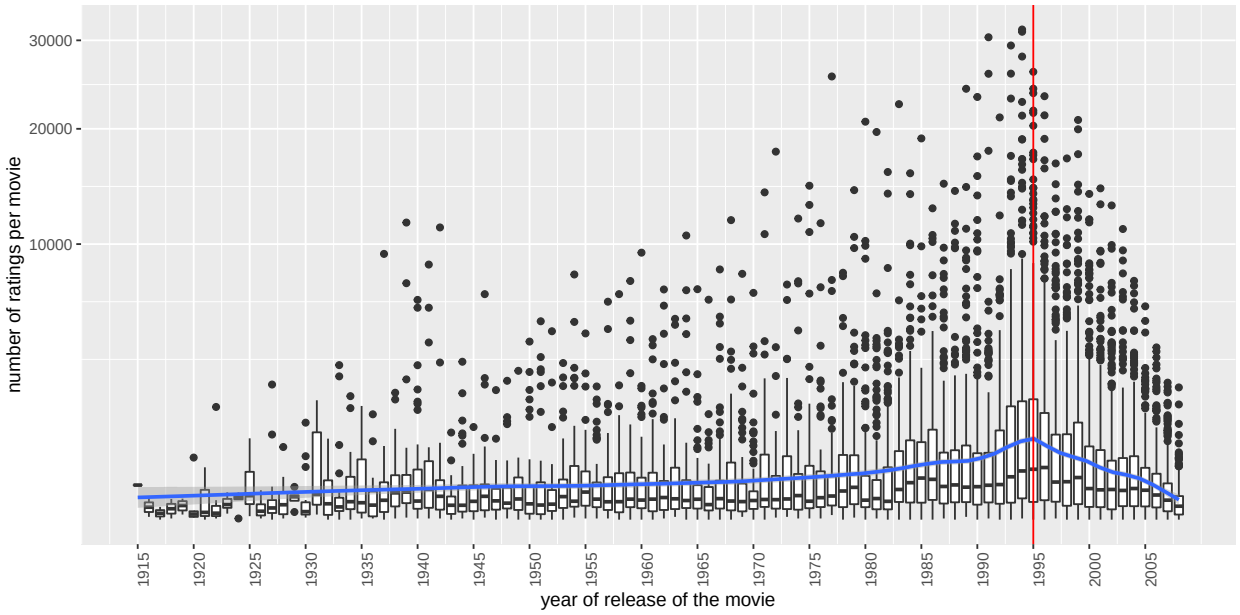
```
edx %>% group_by(movieId) %>%
  summarize(rating = mean(rating), nratings = round(n(), digits = -3)) %>%
  ungroup() %>%
  group_by(nratings) %>%
  summarize(se = sd(rating)/sqrt(n()), rating = mean(rating)) %>%
  ggplot(aes(x = nratings,
             y = rating,
             ymin = rating - 2*se,
             ymax = rating + 2*se)) +
  geom_point() +
  geom_errorbar() +
  geom_smooth(method = "lm")
```



Intuitively, this makes sense since good movies are recommended more often and thus viewed and rated more often. However, if movies are available for a longer period of time, they might accumulate more views and thus potentially receive more ratings. This is confirmed by the following plot, where we can observe that the total number of ratings drops since the year 1995, which is the year of the earliest entry in the `edx` data set, which is indicated by a red line. Additionally, we assume that very old movies, like the ones before 1995, have less ratings because there was no platform to rate them available at this time, thus these movies could not accumulate as many ratings as the ones where a rating platform was available.

```
edx %>%  group_by(movieId, year) %>%
  summarize(nratings = n()) %>%
  ungroup() %>%
  ggplot(aes(x = year, y = nratings)) +
  geom_boxplot(aes(group = year)) +
  geom_smooth(method = "loess", span = 0.2, method.args = list(degree=1))+
  scale_y_continuous(trans = "sqrt") +
  scale_x_continuous(breaks = seq(min(edx$year), max(edx$year), 5)) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  geom_vline(xintercept = year(min(as_datetime(edx$timestamp))), color = "red") +
```

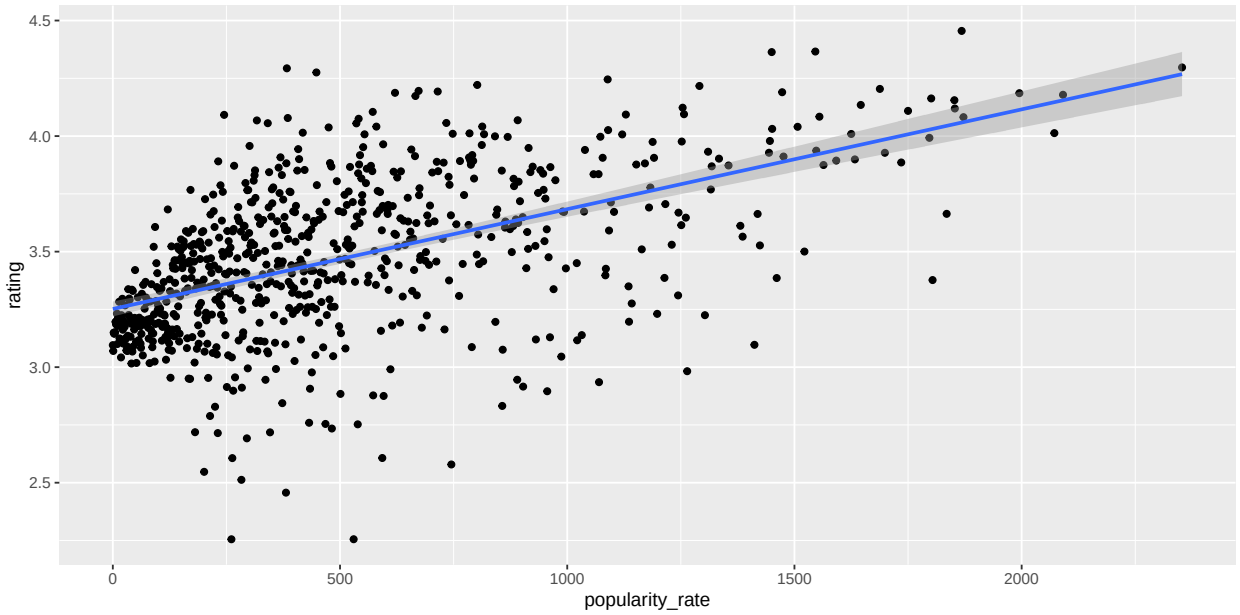
```
  labs(y = "number of ratings per movie", x = "year of release of the movie")
```



Therefore, we should consider the rate of ratings per year $p(i)$ per movie i ("popularity rate") instead of the total number of ratings and observe in the following plot that there is a so-called *Popularity effect* $b_{p(i)}$, which associates movies with high popularity rate to higher ratings.

```
max_year <- max(year(as_datetime(edx$timestamp)))

edx%>%
  group_by(movieId) %>%
  summarize(n = n(),
            years = max_year - year,
            rating = mean(rating)) %>%
  slice(1) %>%
  mutate(popularity_rate = n/years) %>%
  mutate(popularity_rate = round(popularity_rate)) %>%
  group_by(popularity_rate) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(popularity_rate, rating)) +
  geom_point(aes(popularity_rate, rating)) +
  geom_smooth(method = "lm")
```



2.3.5 Genre effect

So far, we have not examined the genre combinations of each movie and its effect on the rating of a movie.

```
# Number of different genre combinations appearing in the edx dataset
length(unique(edx$genres))
```

```
## [1] 797
```

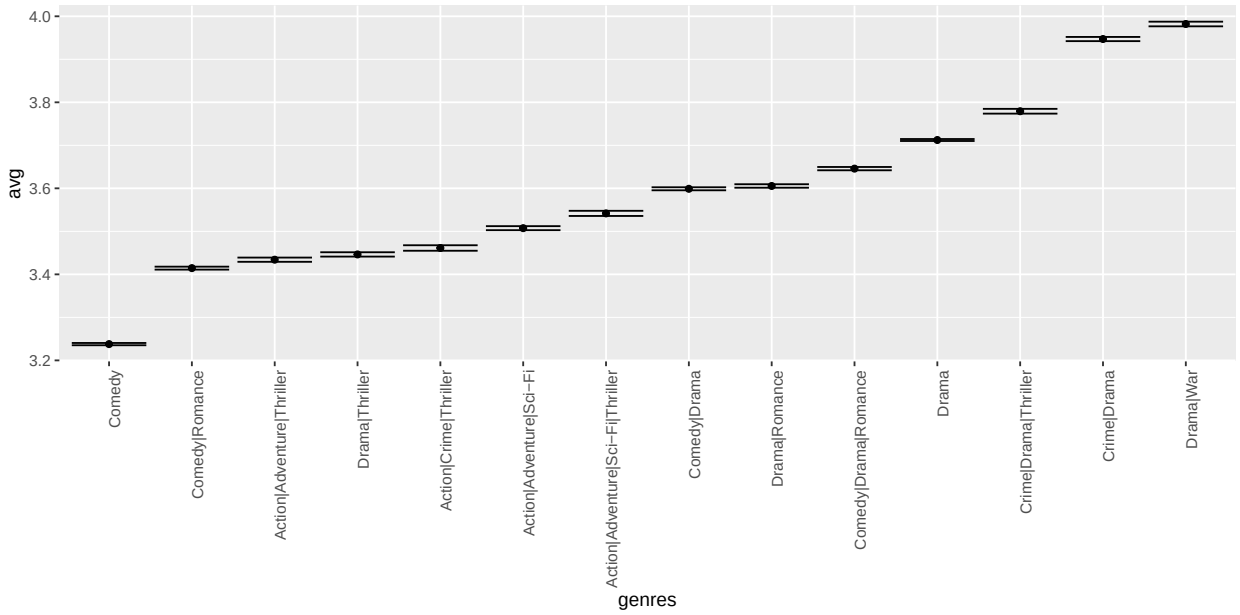
If we filter for genre combinations with more than 10^5 ratings, we end up with 14 genre combinations which cover about 41% of the data:

```
edx %>% group_by(genres) %>%
  summarize(n = n()) %>%
  filter(n >= 100000) %>%
  ungroup() %>%
  summarize(number_of_genres = n(), percent_covered = sum(n)/nrow(edx))
```

```
## # A tibble: 1 x 2
##   number_of_genres percent_covered
##              <int>           <dbl>
## 1               14           0.413
```

Plotting these genre combinations reveals a *Genre effect*.

```
edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 100000) %>%
  mutate(genres = reorder(genres, avg)) %>%
    ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
    geom_point() +
    geom_errorbar() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



2.4 Final model

Our final model is

$$Y_{u,i} = \text{truncate}(\mu + \mu_{u,i} + b_i + b_u - b_{p(i)}) + \epsilon_{u,i},$$

where the notation in the following table is used.

$Y_{u,i}$	rating of user u of movie i
truncate	function that truncates values that are outside the range of ratings
μ	average rating over all movies and all users
$\mu_{u,i} = f(w_{u,i})$	models how average rating changes each week
f	a smooth function
$w_{u,i}$	week of the rating of user u of movie i
b_i	Movie effect bias
b_u	User effect bias
$p(i)$	popularity rate of movie i
$b_{p(i)}$	correction term due to popularity rate effect
$\epsilon_{u,i}$	error term

Notice that the Genre effect is not appropriately incorporated into the model. This is due to limited computational capacity, see below in the section about limitations. In the following, it is described how we estimated the model parameters.

2.4.1 Estimating the Movie and User effect using Regularization

The straightforward way to estimate the model parameters b_i and b_u is to use a linear model `lm`. The computational effort to fit `lm` would be too large. Moreover, `lm`would not account for the fact that some movies have few ratings and some users only rated few movies. Thus we want to penalize $\hat{b}_i$ and $\hat{b}_u$ if they are based on few data points. To do so, regularization is used, i.e. b_i and b_u are estimated by minimizing

$$\hat{Y}_{u,i} = \frac{1}{N} \sum_{u,i} (Y_{u,i} - \mu - b_i - b_u)^2 + \lambda \left(\sum_i b_i^2 + \sum_u b_u^2 \right)$$

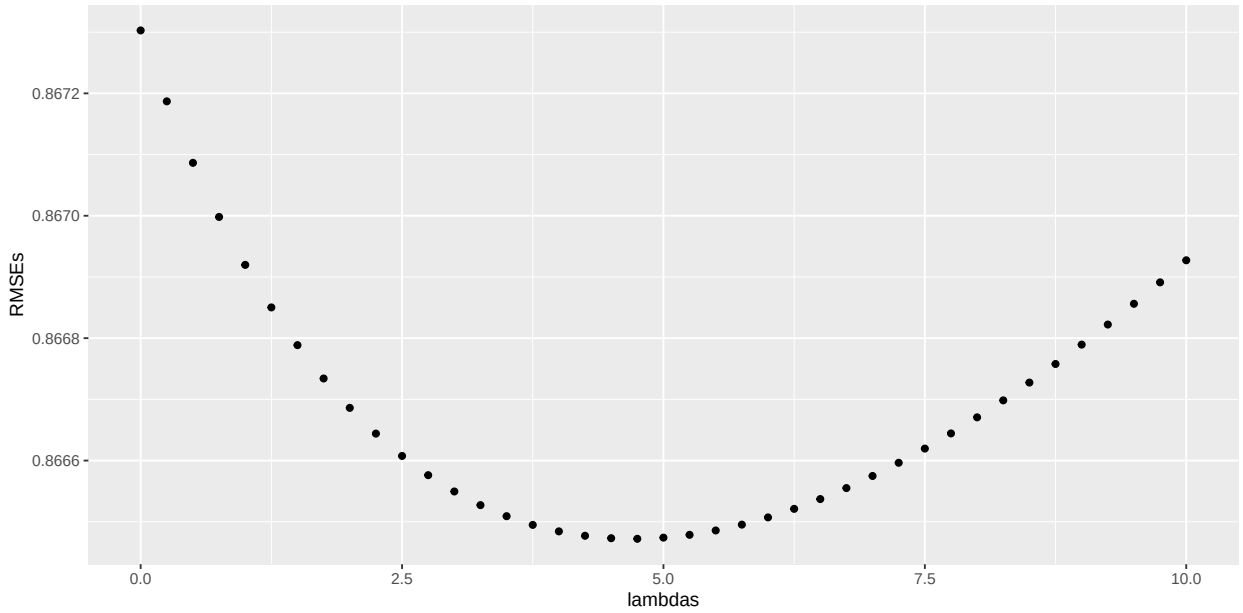
for a given penalizing parameter λ, where N is the total number of observations. Hence

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

and

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

where n_i (resp. n_u) denotes the number of ratings of movie i (resp. user u). To adjust the penalizing parameter λ, we run a 1-fold cross-validation. Its result is shown in the following plot.



Having adjusted lambda, $\hat{b}_i$ and $\hat{b}_u$ can be calculated using the following code:

```
movie_effect <- edx %>% group_by(movieId) %>%
  summarize(b_i_hat = sum(rating - mu_hat)/(n()+lambda))

user_effect <- edx %>%
  left_join(movie_effect, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u_hat = sum(rating - b_i_hat - mu_hat)/(n()+lambda))
```

2.4.2 Estimating the Time effect on average rating

To model the Time effect $\hat{\mu}_{u,i}$ on average rating, a weighted regression is used. To do so, a new variable `date` is introduced, which rounds the `timestamp` of each user's rating to the nearest week using the following code:

```
edx_weeks <- edx %>% mutate(mu_hat = mu_hat) %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  mutate(mu_u_i_hat = rating - mu_hat - b_u_hat - b_i_hat) %>%
  mutate(date = as_datetime(timestamp)) %>%
  mutate(date = round_date(date, unit = "week")) %>%
  group_by(date) %>%
```

```
  summarize(mu_u_i_hat = mean(mu_u_i_hat)) %>%
  mutate(date = as.numeric(date))
```

Now a `loess` model can be fit to calculate $\hat{\mu}_{u,i}$ from `date`. We refer to this model as `time_effect_loess`.

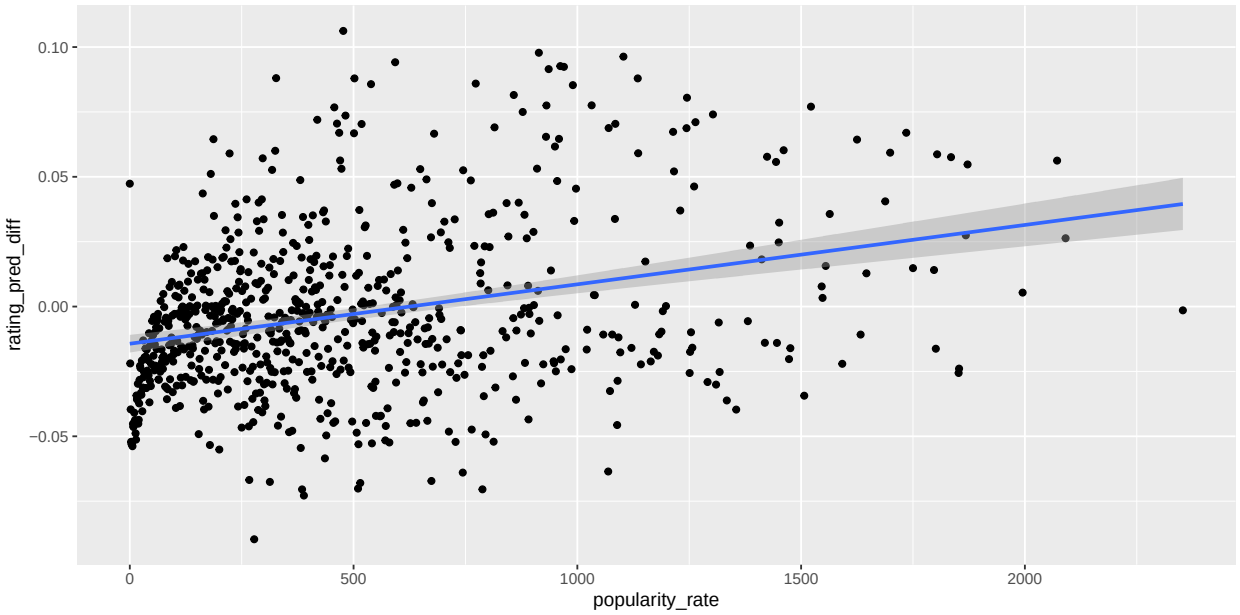
```
set.seed(1, sample.kind="Rounding")
grid <- expand.grid(span = seq(0.15, 0.65, len = 10), degree = 1)
time_effect_loess <- train(mu_u_i_hat ~ date,
                      data = edx_weeks,
                      method = "gamLoess",
                      tuneGrid = grid)
```

2.4.3 Estimating the Popularity effect

Notice that we need to be careful when estimating the Popularity effect since it could be incorporated implicitly into our model. Indeed,we modeled the Movie effect using regularization. Thus the Movie effect bias is penalized less for movies with more ratings, which yields that our model may already show a Popularity effect itself. This can be seen from modeling the `edx` data using the Movie, User and Time effect, comparing it to the data itself and plot it against the popularity rate.

```
max_year <- max(year(as_datetime(edx$timestamp)))
edx_date <- edx %>% mutate(mu_hat = mu_hat) %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  mutate(date = as_datetime(timestamp)) %>%
  mutate(date = round_date(date, unit = "week")) %>%
  mutate(date = as.numeric(date))

edx %>% mutate(mu_hat = mu_hat) %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  mutate(mu_u_i_hat = predict(time_effect_loess, edx_date)) %>%
  mutate(y_hat = mu_hat + b_u_hat + b_i_hat + mu_u_i_hat) %>%
  mutate(y_hat = ifelse(y_hat < 0.5, 0.5, y_hat)) %>%
  mutate(y_hat = ifelse(y_hat > 5, 5, y_hat)) %>%
  group_by(movieId) %>%
  summarize(n = n(),
            years = max_year - year,
            y_hat = mean(y_hat),
            rating = mean(rating)) %>%
  slice(1) %>%
  mutate(popularity_rate = n/years) %>%
  mutate(popularity_rate = round(popularity_rate)) %>%
  group_by(popularity_rate) %>%
  summarize(rating_pred_diff = mean(y_hat - rating)) %>%
  ggplot(aes(popularity_rate, rating_pred_diff)) +
  geom_point(aes(popularity_rate, rating_pred_diff)) +
  geom_smooth(method = "lm")
```



The plot above shows how the difference

$$\text{predicted\_rating} - \text{rating}$$

behaves. More precisely, the higher the popularity rate, the more positive is our error. Thus we already over-estimated the Popularity effect. Hence we need to incorporate a correction term $b_{p(i)}$ into our model. The linear model at the end of the following code can be used to produces the desired estimates $\hat{b}_{p(i)}$.

```
max_year <- max(year(as_datetime(edx$timestamp)))

edx_date <- edx %>% mutate(mu_hat = mu_hat) %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  mutate(date = as_datetime(timestamp)) %>%
  mutate(date = round_date(date, unit = "week")) %>%
  mutate(date = as.numeric(date))

edx_weeks <- edx_date %>%
  mutate(mu_u_i_hat = rating - mu_hat - b_u_hat - b_i_hat) %>%
  group_by(date) %>%
  summarize(mu_u_i_hat = mean(mu_u_i_hat)) %>%
  mutate(date = as.numeric(date))

set.seed(1, sample.kind="Rounding")
grid <- expand.grid(span = seq(0.15, 0.65, len = 10), degree = 1)
time_effect_loess <- train(mu_u_i_hat ~ date,
                           data = edx_weeks,
                           method = "gamLoess",
                           tuneGrid = grid)

# Introduce variable "popularity rate"
edx_popularity_rate_diff <- edx %>% mutate(mu_hat = mu_hat) %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  mutate(mu_u_i_hat = predict(time_effect_loess, edx_date)) %>%
```

```
  mutate(y_hat = mu_hat + b_u_hat + b_i_hat + mu_u_i_hat) %>%
  mutate(y_hat = ifelse(y_hat < 0.5, 0.5, y_hat)) %>%
  mutate(y_hat = ifelse(y_hat > 5, 5, y_hat)) %>%
  group_by(movieId) %>%
  summarize(n = n(),
            years = max_year - year,
            y_hat = mean(y_hat),
            rating = mean(rating)) %>%
  slice(1) %>%
  mutate(popularity_rate = n/years) %>%
  mutate(popularity_rate = round(popularity_rate)) %>%
  group_by(popularity_rate) %>%
  summarize(rating_pred_diff = mean(y_hat - rating))

set.seed(1, sample.kind="Rounding")
popularity_effect_lm_correction <- train(rating_pred_diff ~ popularity_rate,
                                         data =edx_popularity_rate_diff,
                                         method ="lm")
```

3 Results

3.1 RMSE

To evaluate the final product, we run our model on the `validation` set.

```
## loss function to evaluate final product
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

## define test set
test_set <- validation

## preprocess test set to be able to apply the model
test_set <- test_set %>% mutate(mu_hat = mu_hat) %>%
  mutate(date = as_datetime(timestamp)) %>%
  mutate(date = round_date(date, unit = "week")) %>%
  mutate(date = as.numeric(date))
test_pop_rate<- test_set %>% group_by(movieId) %>%
  summarize(n = n(),
            years = max_year - year,
            rating = mean(rating)) %>%
  slice(1) %>%
  mutate(popularity_rate = round(n/years)) %>%
  select(-n, -years, -rating)
test_set <- test_set %>% left_join(test_pop_rate, by = "movieId")

## make predictions
predictions_all <- test_set %>% mutate(mu_hat = mu_hat) %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  mutate(mu_u_i_hat = predict(time_effect_loess, test_set)) %>%
  mutate(b_p_i_hat = predict(popularity_effect_lm_correction, test_set)) %>%
```

```
  mutate(y_hat = mu_hat + b_u_hat + b_i_hat + mu_u_i_hat - b_p_i_hat) %>%
  mutate(y_hat = ifelse(y_hat < 0.5, 0.5, y_hat)) %>%
  mutate(y_hat = ifelse(y_hat > 5, 5, y_hat))

predictions <- predictions_all %>% pull(y_hat)

## evaluate predictions
RMSE(predictions, test_set$rating)
```

```
## [1] 0.8649033
```

We see that RMSE is OK, but could certainly be improved further. A discussion of that can be found below in the section called limitations and future improvements.

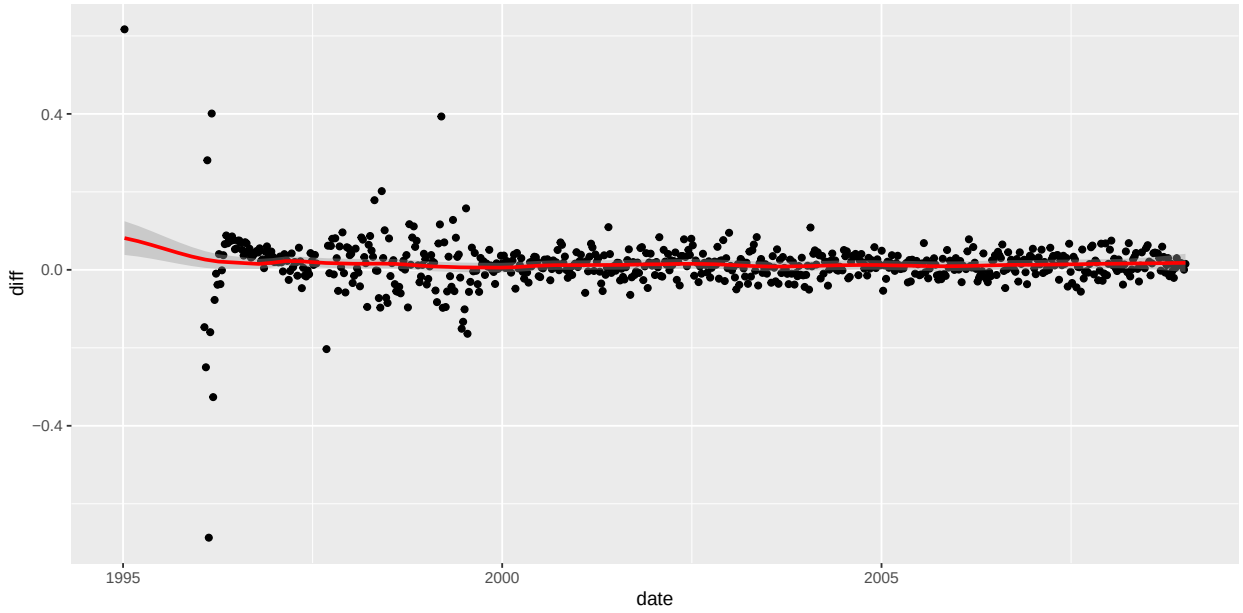
3.2 Evaluation of model predictions

We compare the model's predictions to the true ratings of the `validation` test set. For that we define the difference `diff` by

$$\text{diff} = \text{prediction} - \text{true\_rating}.$$

3.2.1 Time effect

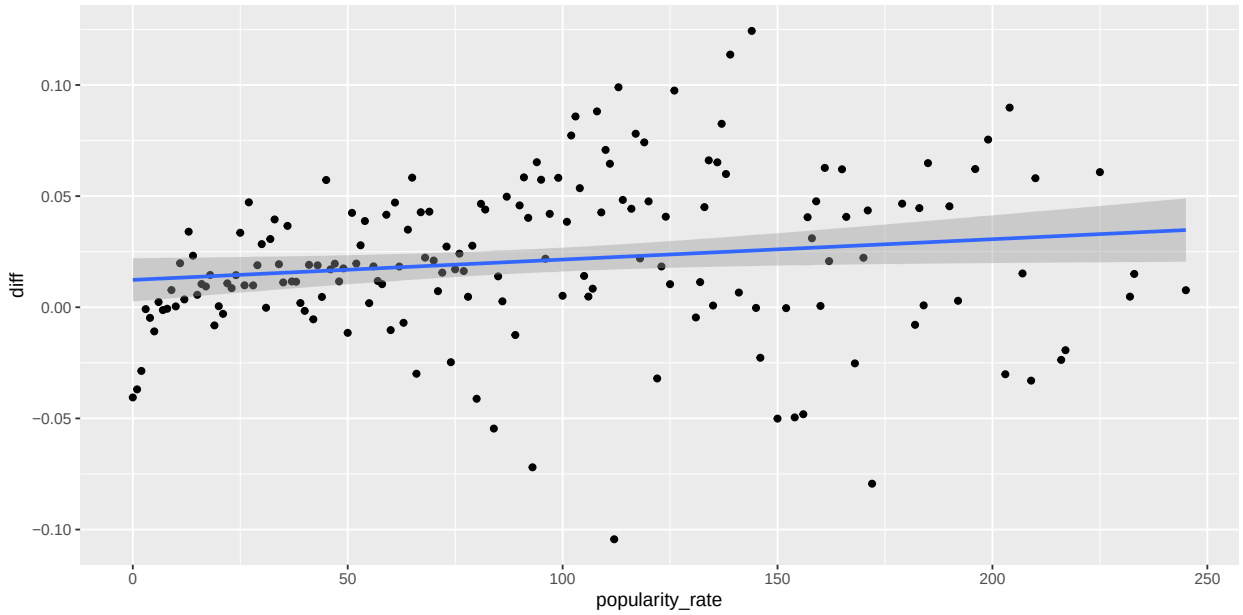
To see whether the time effect was correctly modeled, the difference `diff` is plotted against the week in which the rating was given.



In comparison to the Time effect we observed in the `edx` data set, it seems like we successfully incorporated the Time effect into our model.

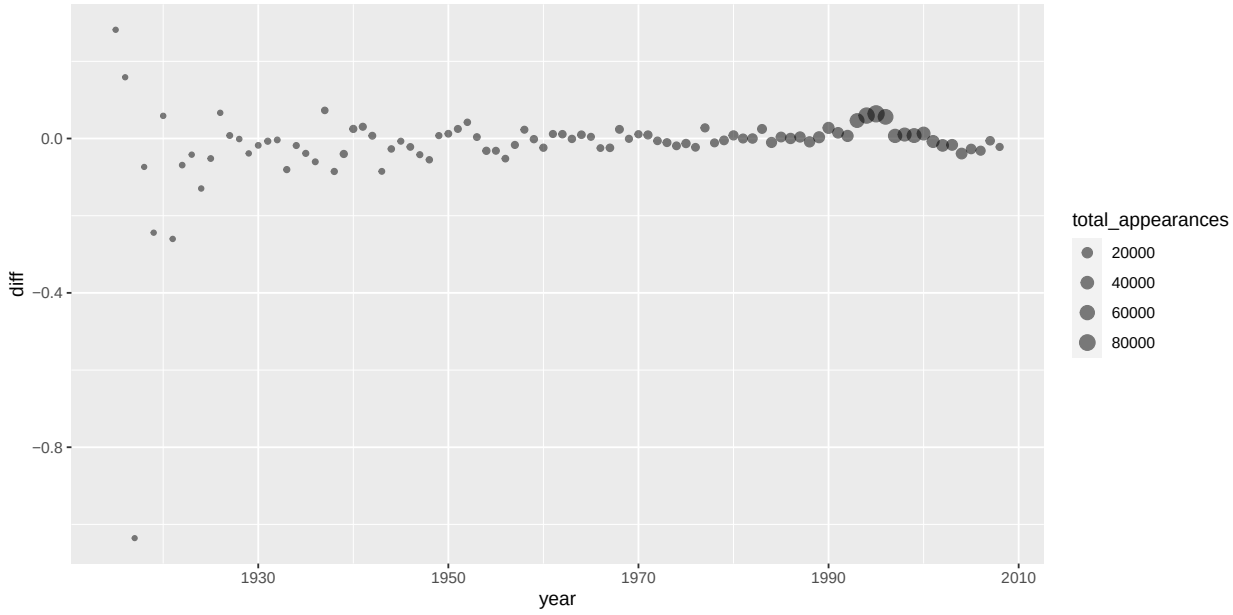
3.2.2 Popularity effect

The following plot shows that we corrected too much of the popularity effect with the summand $b_{p(i)}$ since it is still present in the difference `diff`. Indeed, if we re-run our model without this summand, a RMSE of only 0.8647923 is obtained on the `validation` set (instead of 0.8649033). However, as the `validation` set was the final test set, we are not allowed to fit our model further to this data, so we cannot drop the summand $b_{p(i)}$.



3.2.3 Examine diff and year

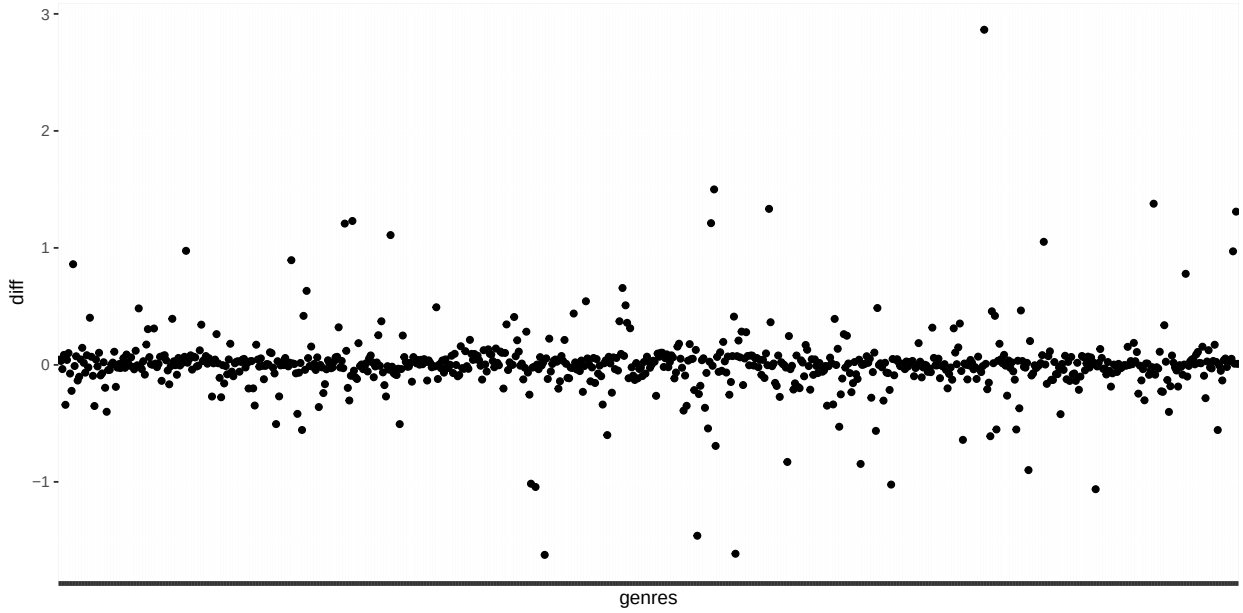
Let us plot the difference `diff` against the year a movie was released.



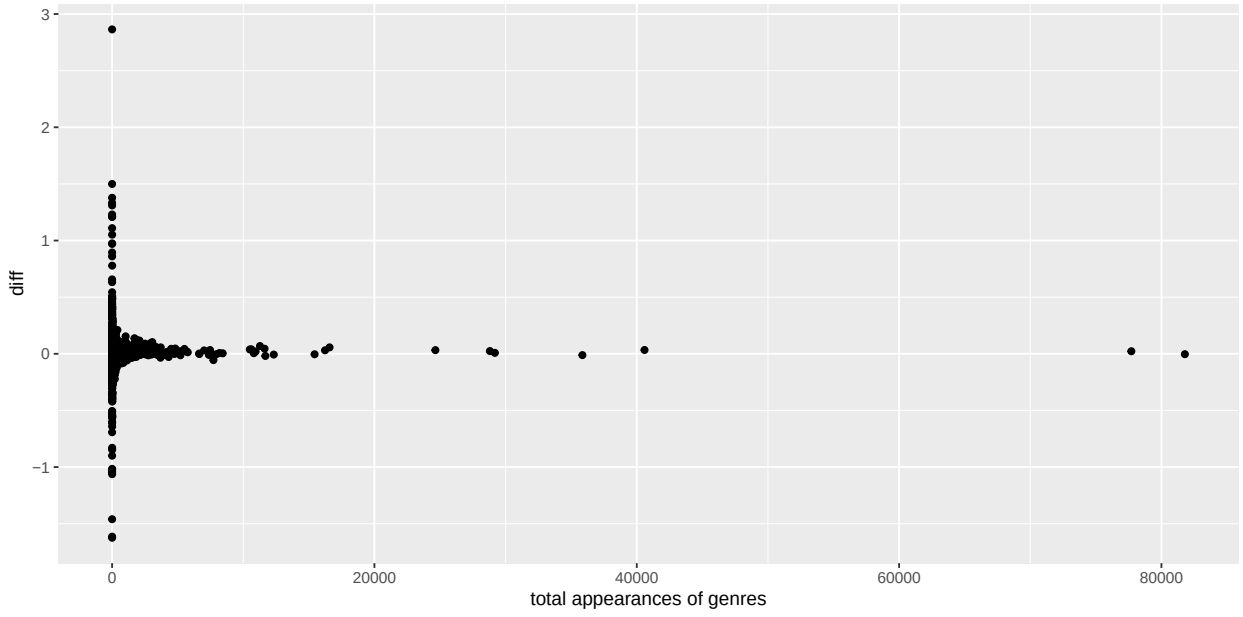
Although is seems that the absolute value of the difference grows as we go back in time, the plot also shows that this actually results from a confounding effect: The older the year of release, the less data points we have. Thus it is harder to predict them, which is a general effect.

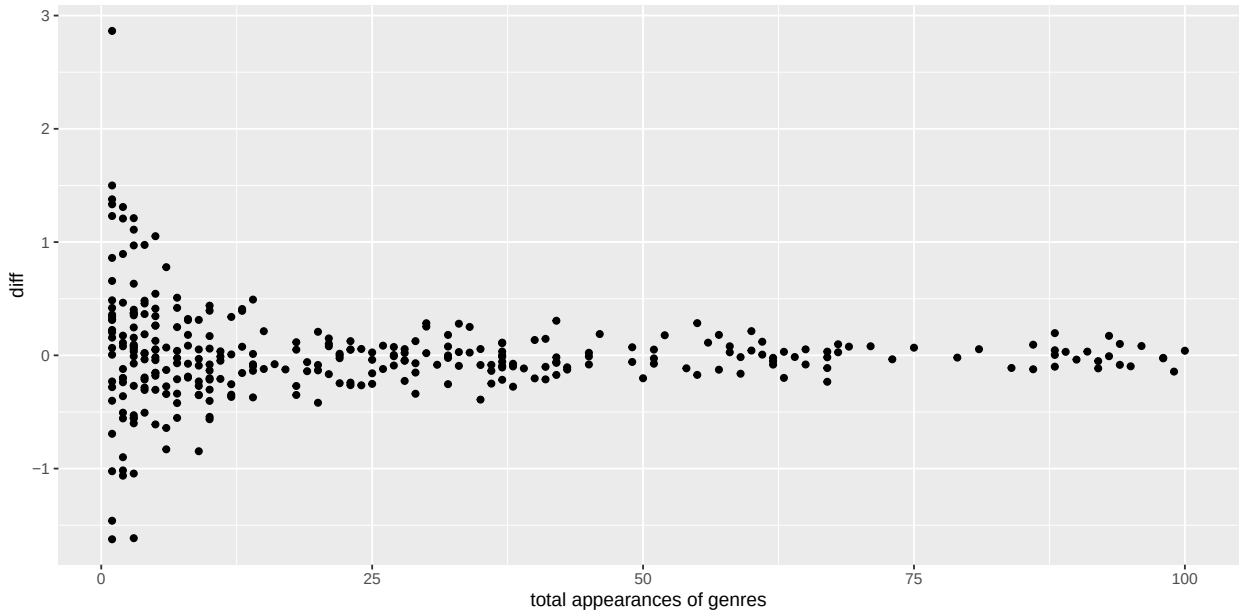
3.2.4 Examine diff and genre

Observe from the following plat, that, for most genres, our predictions are good.



Again, the genres which yield the highest errors are the ones we have the fewest data points available, as the next two plots show.





4 Conclusion

4.1 Summary

Although our model did not make use of factorization techniques and we over-corrected the Popularity effect, it can be used to make acceptable predictions. The improvements through using regularization techniques outweighed the poor performance of the corrected Popularity effect.

4.2 Limitations and future improvements

The strongest limitation throughout the project was computing capacity. In the following, we list starting points for improving the project and give some ideas of how to implement them.

4.2.1 Cross-vaildation

Limited computing capacity forced to use cross-validation with few folds only.

4.2.2 Linear models

Limited computing capacity did not allow us to run a linear model `lm` like the following, which would most likely result in an acceptable estimate of ratings. Instead dealing with the Popularity effect was cumbersome.

```
lm(rating ~ as.factor(movieId)+ as.factor(userId) + popularity_effect, data = edx)
```

4.2.3 Factorization

If we assume that, in addition to the Genre effect, there are groups of users that like one genre more than the other, then using factorization would be the right way to account for this. However, limited computing capacity prevented us from calculating the singular value decomposition needed. The code to do this looks like the following:

```
# Calculate residues
edx_residue <- edx %>% mutate(mu_hat = mu_hat) %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
```

```
  mutate(mu_u_i_hat = predict(time_effect_loess, test_set)) %>%
  mutate(b_p_i_hat = predict(popularity_effect_lm_correction, test_set)) %>%
  mutate(y_hat = mu_hat + b_u_hat + b_i_hat + mu_u_i_hat - b_p_i_hat) %>%
  mutate(y_hat = ifelse(y_hat < 0.5, 0.5, y_hat)) %>%
  mutate(y_hat = ifelse(y_hat > 5, 5, y_hat)) %>%
  mutate(residue = rating - y_hat) %>%
  select(userId, movieId, residue)

edx_residue <- edx_residue %>% spread(key = movieId, value = residue) %>%
  as.matrix()

# Replace NAs with zeros (not a good approximation, but a simple one)
edx_residue[is.na(edx_residue)] <- 0

# Run a SVD to calculate
# residue = U*D*V^t = u_1*d_1*v*1 + u_2*d_2*v*2 + ...
s <- svd(edx_residue[, -1], nu = 1, nv = 1)
```