MovieLens Project

HarvardX PH125.9x - Data Science: Capstone

Emanuela Pannini

2024-10-21

Contents

Introduction

This project aims to predict the rating that an user will give to a particular movie, based on the main general features. In this case all the data came from the Movielens dataset. Due to performances, we will use the version containing just 10M rows and not the full one. The Movielens dataset has the following structure, with 9000055 rows and 10 columns, where these are the original ones:

- `userId`: ID of the user that has rated the movie
- `movieId`: ID of the movie
- `rating`: rate that was assigned to the movie. This is a number between 0.5 and 5, with steps of 0.5
- `timestamp`: the unix time related to the review time
- `title`: the name of the movie and the year of release, embraced between parenthesis
- `genres`: the list of genres assigned to that movie. Each category is divided by | and each movie can be assigned to more than one

and, as it will be clarified in this analysis, these are the columns that we added:

- `releaseYear`: year of release of the movie, extracted by the title column. Before adding this new column it was also checked that each `movieId` has a release year and that each movie has only 1 `releaseYear`
- `reviewDate`: conversion of the `timestamp` column to an human-readable date format
- `deltaReleaseYear`: difference (in years) between the `releaseYear` of the first movie in the dataset and the `releaseYear` of the given movie. The reason why this column is needed is explained later in the document
- `deltaReviewDate`: difference (in weeks) between the `reviewDate` of the first review in the dataset and the `reviewDate` of the given review. As before, further details will be added below

From Edx, it was provided a code that splits the initial dataset into `edx` and `final_holdout_test`. The latter one, will be used only at the end to evaluate the performances of the final model through RMSE, that evaluates the deltas as stated in [1]:

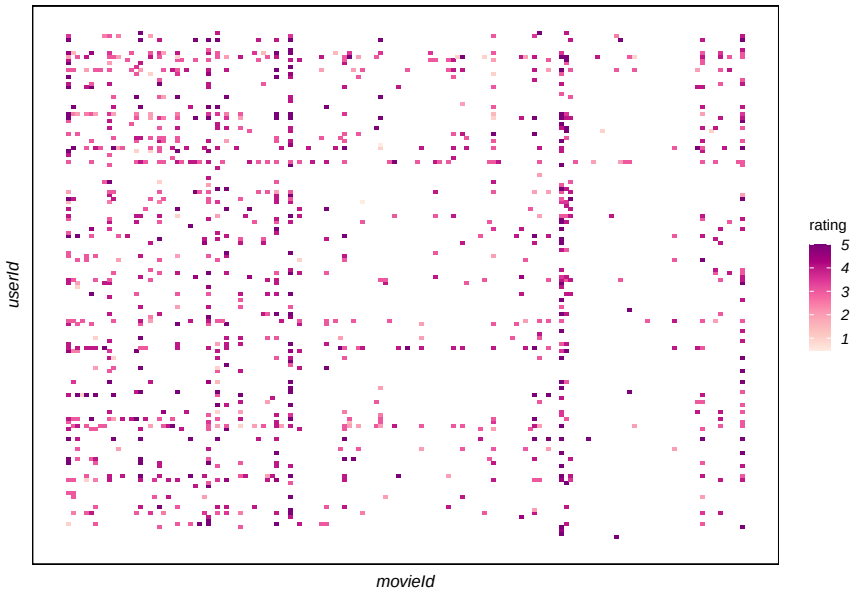
$$RMSE = \sqrt{\frac{1}{N}\sum_{m,u}(\hat{y}_{m,u} - y_{m,u})^2}$$

where $\hat{y}_{m,u}$ is the prediction for movie m and user u and $y_{m,u}$ the actual value.

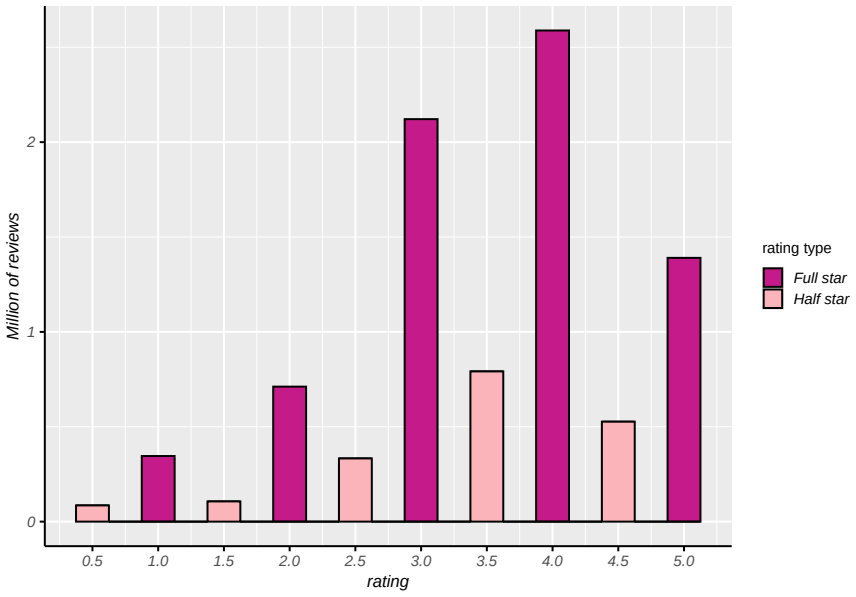
Similarly, the `edx` dataset has been split into `edx_train` and `edx_test` and, in the following sections, we will show the RMSE between `edx_train` and `edx_test`. We will then evaluate the RMSE on the `final_holdout_test` using the best model. For performance reasons, cross-validation or other techniques that perform multiple splitting of the set are not presented here, even though they could be used to test additional cases.

Analysis

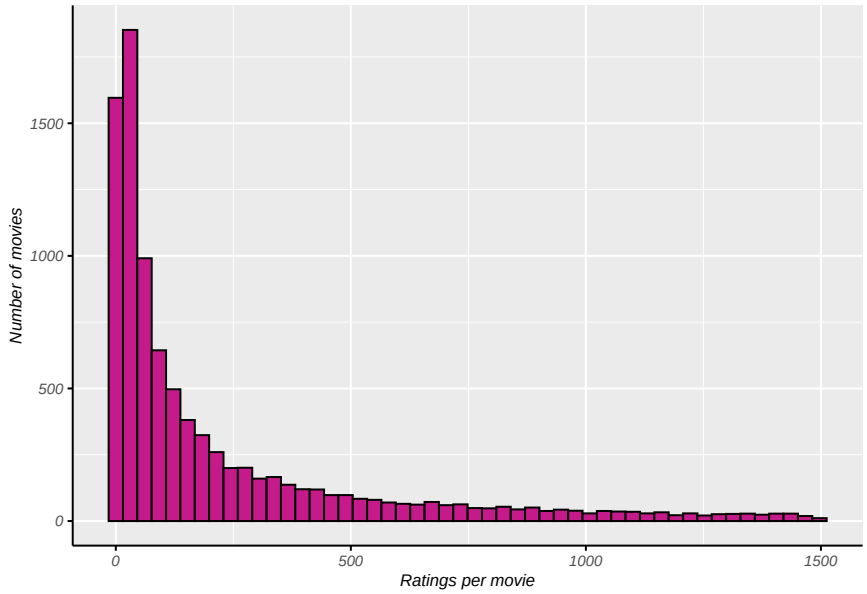
A quick inspection reveals that the Movielens dataset has 10677 distinct movies and 69878 different users. So, as expected, each user has only rated a few movies and each movie has only been rated by a few users. The heatmap that shows whether a user has rated a movie is very sparse, with a fill level of 0.012063 and, since the full heatmap would not be informative, only a section is depicted below:



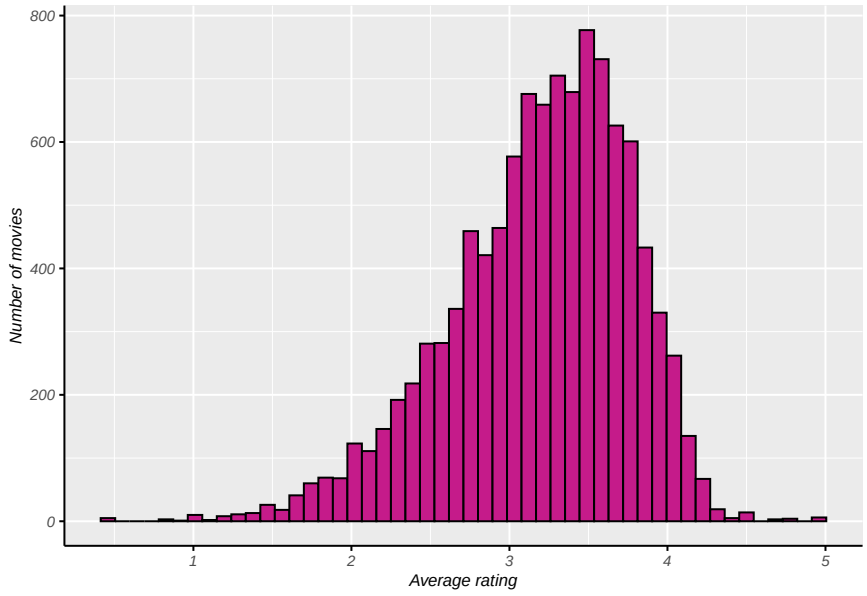
In addition to the one in [1], this heatmap also illustrates the ratings assigned by users to the same `movieId`, with the different ratings highlighted in distinct colours. Viceversa, it is also possible to observe instances where an user has rated different movies, assigning different ratings. Furthermore, as you might expect, users are more likely to give a movie a full star than a half star, with a significant difference between the two categories:



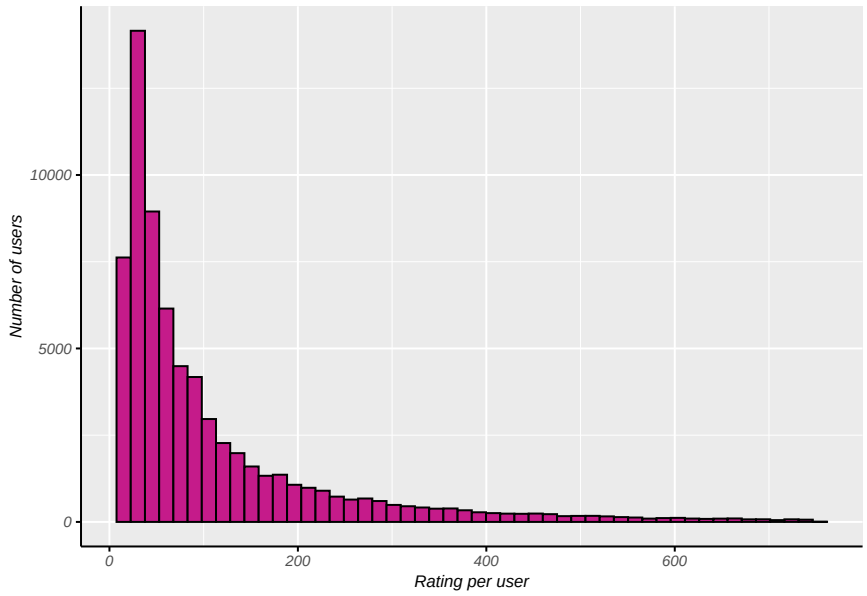
In order to understand what might be the best features to include in the final model, an exploratory analysis is presented below, which should highlight some dependencies. Some of the following plots were inspired by those found in [1], although they were all subsequently redeveloped and adjusted to highlight some features or trends. As expected, one of the most important features is the `movieId`, as it allows us to distinguish between masterpieces and minor works. Clearly some movies received more reviews than others, here are all the movies with less than 1500 reviews, as they are the majority:



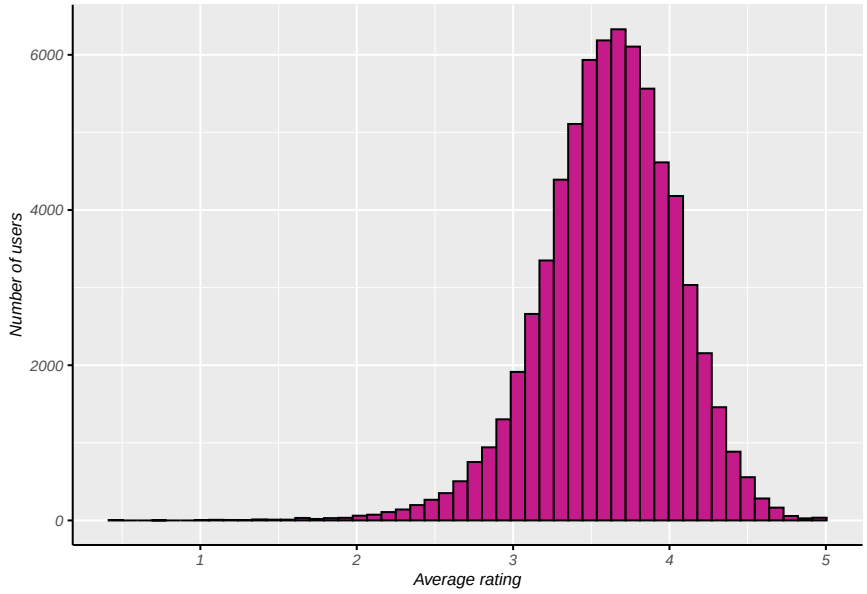
If we instead look at the average rating for each movie, we get the following plot:



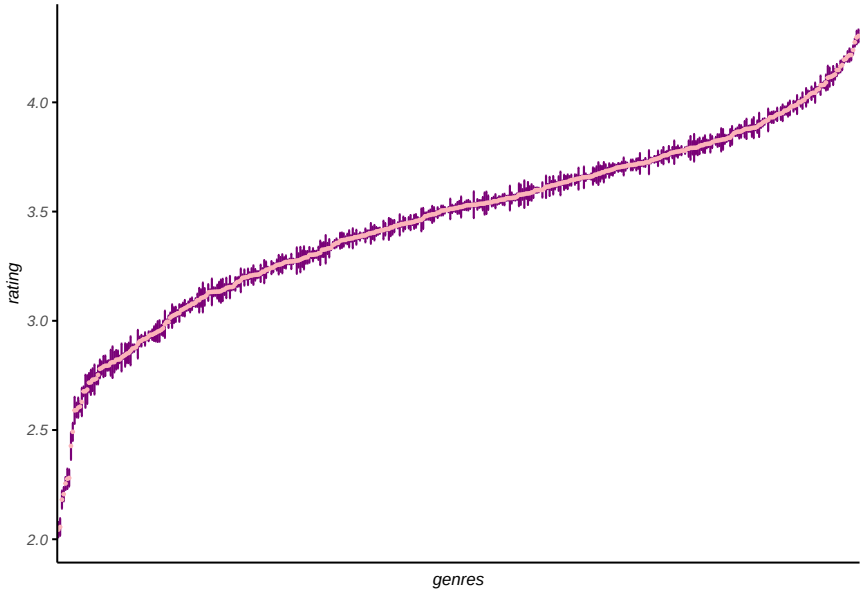
and, as might be expected, the majority of movies received a rating in the middle part of the range, decreasing towards the extremes. The same idea can be extended to users to show the number of reviews given by an user. Also in this case, in order to present an informative plot, the one below has been pre-filtered on the `userId` who have rated less than 750 movies, as they are the great majority:



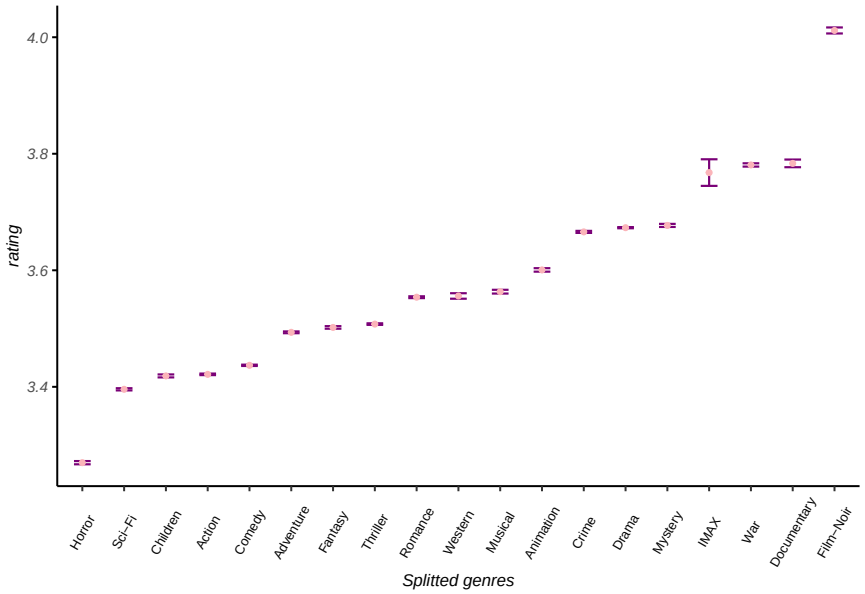
Similarly, below is displayed the average rating given by users:



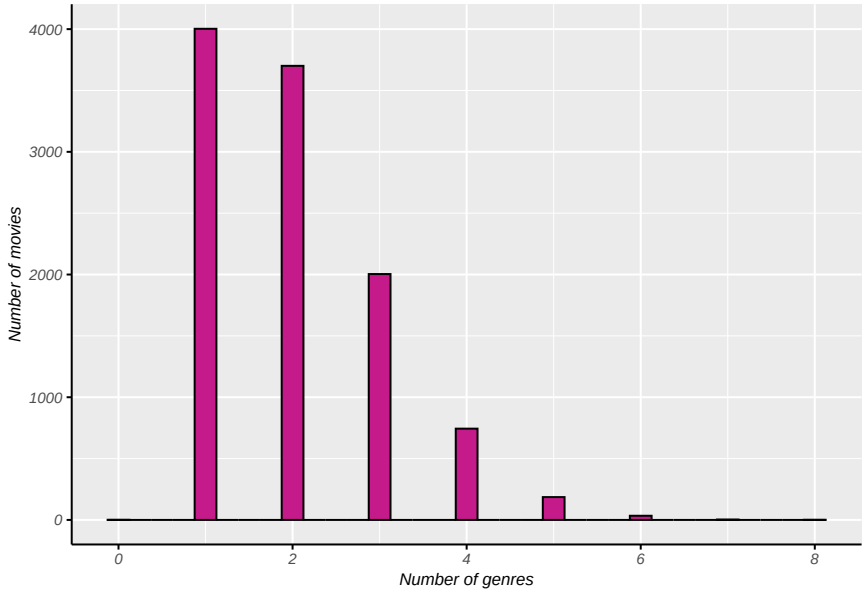
In addition, we intuitively expect the genre to affect the ratings and we developed some plots to verify this hypothesis and to get a better understanding. For example, the first idea was to take the mean and standard error for each category, sort them by mean and highlight the error bars. The name of the category has been removed for readability, but it remains clear that `genres` feature affects the rate and should be included in the models:



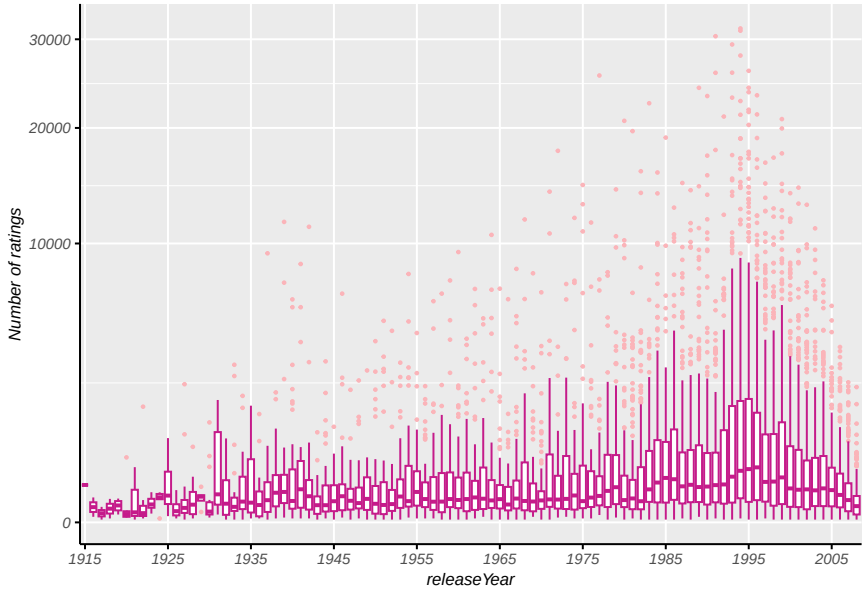
Another hypothesis might be that it is possible to achieve better results by dividing the `genres` into macro-categories, but the result is less informative, as shown in the plot below:



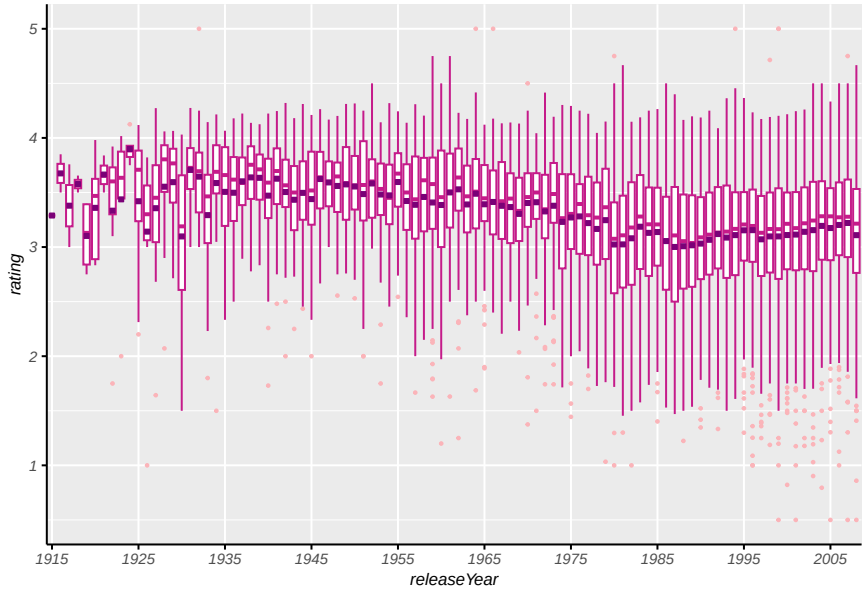
This phenomenon can also be explained by the fact that each movie usually has more than one macro-category, and that this combination tells more about the movie and the expected rating:



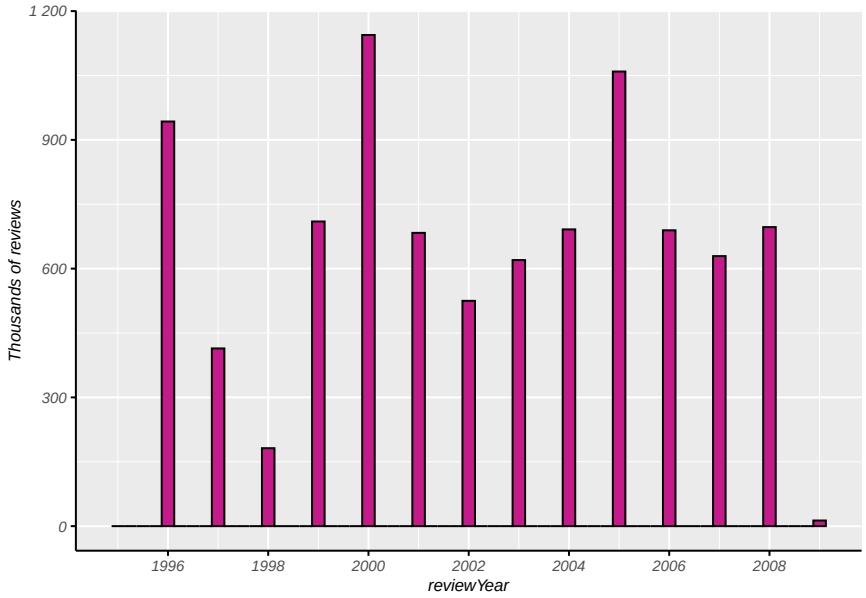
The plot above does not take into account the fact that some `genres` are correlated or excluded due to the definition of the category. For example, it is unlikely to have a movie with `Horror` and `Children` as a category and, on the contrary, it is likely to have `Animation` and `Children` as a combination. It is also possible to perform a deep similarity analysis between categories and genres, however it was not investigated more as it was considered beyond the scope of this project. As anticipated in the previous section, it is also reasonable to predict some differences according to the year of release, and that's why the column `releaseYear` was added by parsing the information in the title. In the plot below, it can be seen how the number of reviews varies over the `releaseYear`, using a standard boxplot showing the outliers:



Of course, it is possible to follow this path and analyse how the rating varies, as shown in the following boxplot structure (i.e. median, quartiles, outliers) and through the purple squares, representing the mean:



Furthermore, we can also note how the number of reviews varies over the `reviewDate` years, starting from the first one on 1995:



This plot could be modified to make it more informative, taking into account the number of movies released during the year and the number of reviews in relation to previous years, scaled over time. Unfortunately, the dataset has a lot of movies released before the 90s, so it is not straightforward to extrapolate how the number of reviews per year affects the rating, even if the plot below attempts to do so. Specifically, the `edx` dataset was pre-filtered on movies released after 1995 and, assuming that each film could be reviewed until the last `reviewDate` in the dataset, the average number of reviews per year was evaluated as the ratio between the total number of reviews and the years after release. So, a smooth fit was performed and it is possible to catch how a higher number of reviews suggests a higher rate:

Nowadays, as the number of reviews has increased through different streaming platforms, a new dataset would have many more recent reviews and it will also be possible to analyse this trend over a longer period of time. Furthermore, one could also better verify if and how users tend to review masterpieces even a long time after their release.

Models

Since we want to predict ratings, we expect that the simplest idea should at least include some of the features that have emerged in this brief analysis (i.e. movies, users, genres and review week). So here we follow and extend the general idea proposed in [1], using a linear estimation of the target (due to the same limitations related to machine capability). Already knowing that these simpler models will not fit sufficiently, the first model is:

$$Y_{u,m} = \mu + b_m + b_u + \sum_k x^k_{u,m} b_g + b_y + \varepsilon_{u,m}$$

where u refers to the user, m to the movie, $x^k_{u,m}$ equal 1 if and only if the couple u, m has genre k and y is the year of release. In this way it is possible to assign a different weight to each specific genre, as shown also in the previous section, and also to keep track of the historical period of the production.

From a technical point of view, even if it is not necessary, in this report all models will be explained by functions in order to get a better visualisation. Moreover, unless otherwise specified, the training part on `edx_train` and the test part on `edx_test` will be kept together in order to have a complete overview. In this case, the code for this simpler model is as follows:

```
## simple_prediction <- function(){
##
##   # Train the model on the edx_train set
##   mu <- mean(edx_train$rating)
##   movie_effect <- edx_train %>%
##     group_by(movieId) %>%
##     summarize(b_m = mean(rating - mu))
##   user_effect <- edx_train %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     group_by(userId) %>%
##     summarize(b_u = mean(rating - mu - b_m))
##   genre_effect <- edx_train %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     left_join(user_effect, by = 'userId') %>%
##     group_by(genres) %>%
##     summarize(b_g = mean(rating - mu - b_m - b_u))
##   year_effect <- edx_train %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     left_join(user_effect, by = 'userId') %>%
##     left_join(genre_effect, by = 'genres') %>%
##     group_by(releaseYear) %>%
##     summarize(b_y = mean(rating - mu - b_m - b_u - b_g))
##
##   # Test the model on the edx_test set
##   predicted_ratings <- edx_test %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     left_join(user_effect, by = 'userId') %>%
##     left_join(genre_effect, by = 'genres') %>%
##     left_join(year_effect, by = 'releaseYear') %>%
##     mutate(prediction = mu + b_m + b_u + b_g + b_y,
##            rounded_prediction = sapply(prediction, round_predictions))
##
##   # The function returns the standard RMSE and the rounded RMSE
##   RMSE_standard <- RMSE(predicted_ratings$prediction, edx_test$rating)
```

```
##   RMSE_rounded <- RMSE(predicted_ratings$rounded_prediction, edx_test$rating)
##   return(list(RMSE_standard = RMSE_standard, RMSE_rounded = RMSE_rounded))
## }
```

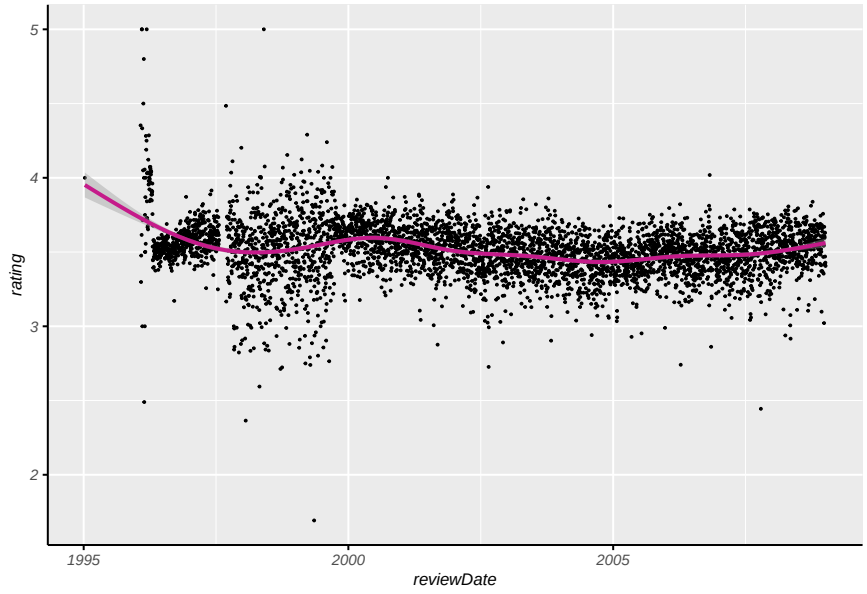
and evaluated on `edx_test` gave a result of as RMSE:

Table 1: RMSEs summary

Model	RMSE_standard	RMSE_rounded
Simple model	0.863307	0.8752221

Since the users could only express ratings with values between 0.5 and 5 in steps of 0.5, we tried to round the prediction to the nearest feasible value before evaluating the RMSE. After doing that we observed that this option results in a higher RMSE, so the prediction is worse. The following models will always report both versions of the prediction to show that this idea always under performs when compared to the non-rounded one.

As the RMSE value is not satisfying, the idea is to increase the accuracy by refining the effect related to the review week. As explained in the Analysis section and as suggested by the exercises in the [1], the plot shows a time dependence of this feature:



So, we will use the `mgcv` library to create a smooth fit and include it in the next model. It was decided to use the `mgcv` library [2] because other techniques (such as `gam`) caused a lot of performance problems on the machine. For the same reason and to avoid overfitting, the `reviewDate` information (originally present in the dataset) was aggregated to a weekly level. At this point, we imagined that both the review week and the release year would be a good fit, so the next model was developed:

```
## double_time_dependence_model <- function(){
##
##   # Train the model on the edx_train set
##   mu <- mean(edx_train$rating)
##
##   # Smooth fit on the deltaReviewDate column and evaluation of the effects
```

```
##   review_week_fit <- gam(rating ~ s(deltaReviewDate, bs = 'cs'),
##                                  family = gaussian(),
##                                  data = edx_train)
##   review_week_effect <- review_week_effects(edx_train, review_week_fit, mu)
##
##   # Smooth fit on the deltaReleaseYear column and evaluation of the effects
##   release_year_fit <- gam(rating ~ s(deltaReleaseYear, bs = 'cs'),
##                                   family = gaussian(),
##                                   data = edx_train)
##   release_year_effect <- release_year_effects(edx_train, release_year_fit, mu)
##
##   movie_effect <- edx_train %>%
##     group_by(movieId) %>%
##     summarize(b_m = mean(rating - mu - review_week_effect[deltaReviewDate+1]
##                          - release_year_effect[deltaReleaseYear+1]))
##   user_effect <- edx_train %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     group_by(userId) %>%
##     summarize(b_u = mean(rating - mu - b_m - review_week_effect[deltaReviewDate+1]
##                          - release_year_effect[deltaReleaseYear+1]))
##   genre_effect <- edx_train %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     left_join(user_effect, by = 'userId') %>%
##     group_by(genres) %>%
##     summarize(b_g = mean(rating - mu - b_m - b_u
##                           - review_week_effect[deltaReviewDate+1]
##                           - release_year_effect[deltaReleaseYear+1]))
##
##   # Test the model on the edx_test set
##   predicted_ratings <- edx_test %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     left_join(user_effect, by = 'userId') %>%
##     left_join(genre_effect, by = 'genres') %>%
##     mutate(prediction = mu + b_m + b_u + b_g +
##              review_week_effect[deltaReviewDate+1] +
##              release_year_effect[deltaReleaseYear+1],
##            rounded_prediction = sapply(prediction, round_predictions))
##
##   # The function returns the standard RMSE and the rounded RMSE
##   RMSE_standard <- RMSE(predicted_ratings$prediction, edx_test$rating)
##   RMSE_rounded <- RMSE(predicted_ratings$rounded_prediction, edx_test$rating)
##   return(list(RMSE_standard = RMSE_standard, RMSE_rounded = RMSE_rounded))
## }
```

The RMSE value is better than the previous one, but it is still not good enough:

Table 2: RMSEs summary

Model	RMSE_standard	RMSE_rounded
Simple model	0.8633070	0.8752221
Double time dependence	0.8632506	0.8752898

From a detailed inspection of the results, it is possible to see that the smooth function is not really suitable when it is also applied to the release year, as it is more related to the movie. So, taking into account all this information, the next idea is to mix these two approaches to finalise this model:

```
## time_dependence_model <- function(){
##
##   # Train the model on the edx_train set
##   mu <- mean(edx_train$rating)
##
##   # Smooth fit on the deltaReviewDate column and evaluation of the effects
##   review_week_fit <- gam(rating ~ s(deltaReviewDate, bs = 'cs'),
##                                     family = gaussian(),
##                                     data = edx_train)
##   review_week_effect <- review_week_effects(edx_train, review_week_fit, mu)
##
##   movie_effect <- edx_train %>%
##     group_by(movieId) %>%
##     summarize(b_m = mean(rating - mu - review_week_effect[deltaReviewDate+1]))
##   user_effect <- edx_train %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     group_by(userId) %>%
##     summarize(b_u = mean(rating - mu - b_m - review_week_effect[deltaReviewDate+1]))
##   genre_effect <- edx_train %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     left_join(user_effect, by = 'userId') %>%
##     group_by(genres) %>%
##     summarize(b_g = mean(rating - mu - b_m - b_u
##                          - review_week_effect[deltaReviewDate+1]))
##   year_effect <- edx_train %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     left_join(user_effect, by = 'userId') %>%
##     left_join(genre_effect, by = 'genres') %>%
##     group_by(releaseYear) %>%
##     summarize(b_y = mean(rating - mu - b_m - b_u - b_g
##                          - review_week_effect[deltaReviewDate+1]))
##
##   # Test the model on the edx_test set
##   predicted_ratings <- edx_test %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     left_join(user_effect, by = 'userId') %>%
##     left_join(genre_effect, by = 'genres') %>%
##     left_join(year_effect, by = 'releaseYear') %>%
##     mutate(prediction = mu + b_m + b_u + b_g + b_y
##                         + review_week_effect[deltaReviewDate+1],
##            rounded_prediction = sapply(prediction, round_predictions))
##
##   RMSE_standard <- RMSE(predicted_ratings$prediction, edx_test$rating)
##   RMSE_rounded <- RMSE(predicted_ratings$rounded_prediction, edx_test$rating)
##   return(list(RMSE_standard = RMSE_standard, RMSE_rounded = RMSE_rounded))
## }
```

In this case, the smooth fit is only applied to the `reviewDate` and the `releaseYear` is considered as a standard effect. This seems to be more consistent with respect to the Analysis section and indeed gives a more accurate view of the real dependencies:

Table 3: RMSEs summary

Model	RMSE_standard	RMSE_rounded
Simple model	0.8633070	0.8752221
Double time dependence	0.8632506	0.8752898
Time dependence	0.8631671	0.8751312

However, not all movies should have the same strength to adjust the prediction [1], so we decided to add a regularisation term to compensate for some discrepancies, resulting in the following model:

```
## regularized_time_dependent_model <- function(current_lambda){
##
##   # Train the model on the edx_train set
##   mu <- mean(edx_train$rating)
##
##   # Smooth fit on the deltaReviewDate column and evaluation of the effects
##   review_week_fit <- gam(rating ~ s(deltaReviewDate, bs = 'cs'),
##                          family = gaussian(),
##                          data = edx_train)
##   review_week_effect <- review_week_effects(edx_train, review_week_fit, mu)
##
##   movie_effect <- edx_train %>%
##     group_by(movieId) %>%
##     summarize(b_m = sum(rating - mu - review_week_effect[deltaReviewDate+1])
##               /(n() + current_lambda))
##   user_effect <- edx_train %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     group_by(userId) %>%
##     summarize(b_u = sum(rating - mu - b_m - review_week_effect[deltaReviewDate+1])
##               /(n() + current_lambda))
##   genre_effect <- edx_train %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     left_join(user_effect, by = 'userId') %>%
##     group_by(genres) %>%
##     summarize(b_g = sum(rating - mu - b_m - b_u - review_week_effect[deltaReviewDate+1])
##               /(n() + current_lambda))
##   year_effect <- edx_train %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     left_join(user_effect, by = 'userId') %>%
##     left_join(genre_effect, by = 'genres') %>%
##     group_by(releaseYear) %>%
##     summarize(b_y = sum(rating - mu - b_m - b_u - b_g -
##                           review_week_effect[deltaReviewDate+1])/(n() + current_lambda))
##
##   # Return the list of parameters after the training phase
##   return(list(
##     mu = mu,
##     movie_effect = movie_effect,
##     user_effect = user_effect,
##     genre_effect = genre_effect,
##     year_effect = year_effect,
##     review_week_effect = review_week_effect,
##     current_lambda = current_lambda
##   ))
```

```
## }
```

In this case, the training and evaluation parts are stored in two different functions to better distinguish the two phases. As you can see from the above code, this model relies on a λ value that should be fitted in an appropriate way to adjust the importance of each entry. So, the main idea is to train on `edx_train` with different λ values, test each model on `edx_test` and choose the best value for λ by selecting the one that minimises the RMSE. The function that we will use to predict the ratings is:

```
## prediction_regularized_time_dependent_model <- function(
##                                         current_regularized_time_dependent_model,
##                                         target_dataframe){
##
##   mu <- current_regularized_time_dependent_model$mu
##   movie_effect <- current_regularized_time_dependent_model$movie_effect
##   user_effect <- current_regularized_time_dependent_model$user_effect
##   genre_effect <- current_regularized_time_dependent_model$genre_effect
##   year_effect <- current_regularized_time_dependent_model$year_effect
##   review_week_effect <- current_regularized_time_dependent_model$review_week_effect
##   current_lambda <- current_regularized_time_dependent_model$current_lambda
##
##   # Test the model on the target set
##   predicted_ratings <- target_dataframe %>%
##     left_join(movie_effect, by = 'movieId') %>%
##     left_join(user_effect, by = 'userId') %>%
##     left_join(genre_effect, by = 'genres') %>%
##     left_join(year_effect, by = 'releaseYear') %>%
##     mutate(prediction = mu + b_m + b_u + b_g + b_y + review_week_effect[deltaReviewDate+1],
##            rounded_prediction = sapply(prediction, round_predictions))
##
##   RMSE_standard <- RMSE(predicted_ratings$prediction, target_dataframe$rating)
##   RMSE_rounded <- RMSE(predicted_ratings$rounded_prediction, target_dataframe$rating)
##   return(list(lambda_value = current_lambda,
##               RMSE_standard = RMSE_standard,
##               RMSE_rounded = RMSE_rounded))
## }
```

In order to tune and choose the optimal value for λ, it is useful to visualise the deltas in RMSE associated with the different values, so the following plot and summary have been constructed:

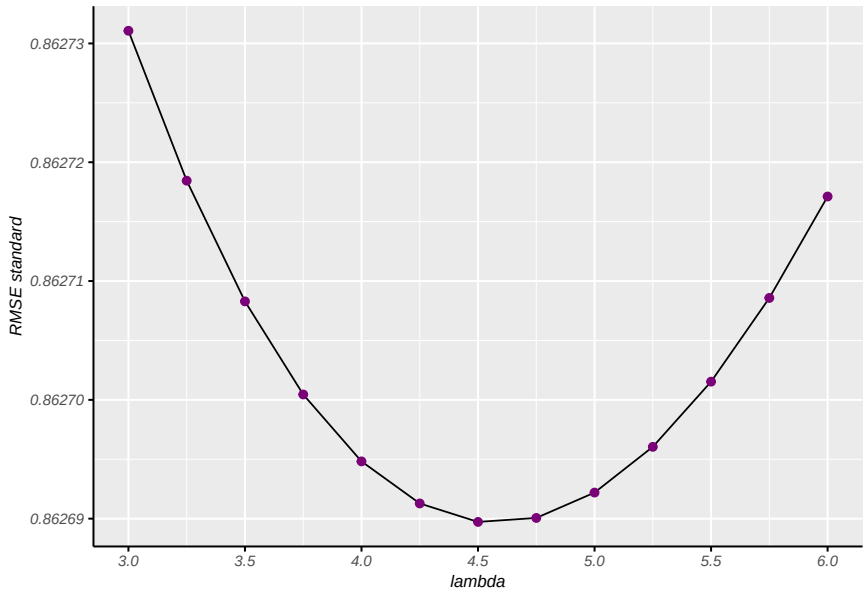


Table 4: Regularized RMSEs summary

lambda_value	RMSE_standard	RMSE_rounded
3.00	0.8627311	0.8746174
3.25	0.8627184	0.8746290
3.50	0.8627083	0.8746378
3.75	0.8627005	0.8746495
4.00	0.8626948	0.8746371
4.25	0.8626913	0.8746449
4.50	0.8626897	0.8745898
4.75	0.8626901	0.8745664
5.00	0.8626922	0.8745468
5.25	0.8626960	0.8745699
5.50	0.8627015	0.8745780
5.75	0.8627086	0.8745982
6.00	0.8627171	0.8745977

For performance reasons, only a subset of the tests for fine-tuning the value of λ is presented in this final version. If you have a better machine and are interested, you can modify the declaration of the `lambdas` variable with a different range, including a finer step and/or a wider range. On the contrary, you can speed up this part by setting the range of `lambdas` to a specific value. Anyway, it is possible to determine 4.5 as the final value for λ and the models presented above can be summarized as follows:

Table 5: RMSEs summary

Model	RMSE_standard	RMSE_rounded
Simple model	0.8633070	0.8752221
Double time dependence	0.8632506	0.8752898
Time dependence	0.8631671	0.8751312
Regularized time dependence	0.8626897	0.8745898

Results

From the Analysis and Models sections it is straightforward to notice that the regularised model that also includes the time dependence is the best one. However, as it is also important to take into account performances, it is important to notice that the tuning part of λ is quite time consuming, especially on standard laptops. On the contrary, the other models are faster but lose some accuracy, even if the results are not so bad if compared to the simplicity of the model. In a real-world scenario, it is also important to consider that the training phase is run once (or few times) and after that only the predictions are needed and, under this point of view, all the models produce an output in a reasonable time.

Anyway, the final model is the time-dependent one with a regularisation term. This approach includes movie, user, genre, year of release and week of review, trying to extrapolate as many features as possible. Thus, the final model applied to the `final_holdout_test` gave an RMSE of:

Table 6: RMSEs final holdout test

Model	lambda_value	RMSE_standard	RMSE_target
Regularized time dependence	4.5	0.8645754	< 0.86490

and it is satisfactory when compared to the project requirements.

Conclusions

This report presents an analysis of the MovieLens dataset, beginning with an examination of its general structure and subsequently exploring the principal connections between its features In this case, the preprocessing phase was not required, as Edx had already provided the code for the dataframe. Nevertheless, this phase is typically a crucial stage in any project. Similarly, no data cleaning was performed, as the data were already well structured and no rows needed to be removed. Furthermore, all the data were coherent and valued as expected, facilitating the inspection and modeling phase. Subsequently, a simple linear estimation was employed, which was then improved with the addition of other meaningful effects, including a time dependence on review week and a regularisation term. The final model gave us a satisfactory RMSE related to the course target, even if it is of course possible to reduce it. From a theoretical point of view it could be a good idea to perform a PCA analysis or some common ML models that can better understand the main features (i.e. random forest, knn, etc. . .). Performance limitations were also considered in this development, so some of them were excluded for this reason. In addition, another idea could be to use cross-validation instead of splitting the `edx` set into fixed training and test sets. It is also possible to develop a similarity analysis to get a better insight into the real impact of the `genres` column. This idea is not presented here because it did not produce a satisfactory result, although it could be a good starting point for extending this project.

Notes

In this project, the impersonal form "we" was employed. However, it should be noted that the project was developed independently, without external interference. As explained during the course and highlighted in [1] and [3], it is important to visualise data in the most readable manner for everyone. So, all the graphical elements in this project are created using the RdPu palette, as you can see in [4]. In addition, all analyses were performed with a laptop with 8GB RAM and i5-1135G7 CPU so they took quite a long time and some other analyses would have been prohibitive. However, if your machine is better than mine, you can run multiple tests on different λ or run additional models. Otherwise, if your machine is not that powerful, I would like to inform you that running the whole project could take more than 2 hours, even after having properly reduced the fine tune part for λ. It might be helpful to execute only the R file, so that you can run only the interesting parts, avoiding useless computations and allowing flexibility in testing.

Bibliography

[1] R. A. Irizarry, *Introduction to data science*. Chapman; Hall/CRC, 2019.

[2] S. Wood, *Mixed GAM computation vehicle with automatic smoothness estimation*. 2023-12-21.

[3] B. Rudis, N. Ross, and S. Garnier, "Introduction to the viridis color maps." Accessed: Sep. 29, 2024. [Online]. Available: https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html#the-color-scales

[4] A. Zeileis and P. Murrell, "Coloring in r's blind spot," *The R Journal*, vol. 15, pp. 240–256, 2023, doi: 10.32614/RJ-2023-071.