HarvardX: PH125.9x Capstone - MovieLens Project

Friederike David

2020-08-04

Contents

This project is part of the HarvardX's Data Science Capstone course, which is the last out of nine courses within the HarvardX's Data Science Professional Certificate.

Introduction

Movie recommendation systems are a common application of machine learning models that aim to predict how a given user would rate a given movie in order to generate recommendations for movies a given user is likely to enjoy. Such recommendation systems are employed for example by streaming services like Netflix to enhance the user experience and therefore well-preforming models can create considerable business value.

In this project, a movie recommendation system based on the MovieLens 10M Dataset is implemented, which predicts movie recommendations for a given movie and user within the dataset from a set of input features. The dataset consists of about 10 million movie ratings with additional information e.g. on movie genre and rating datetime, which are used to develop a machine learning model as detailed in the methods section.

Briefly, as first step the validation subset that is exclusively used for the final model evaluation is split from the input dataset. The remaining data points are explored and cleaned before deciding on a regularized additive model as modeling approach and then further split into a training and test set for model optimization. Finally, model performance is evaluated in the results section based on the validation set. Both for model development and evaluation, the root mean squared error (RMSE) between predicted and true ratings is used as loss function.

Methods

Data input

The publicly available MovieLens 10M Dataset is downloaded and split into a validation set (`validation`, ~10%) and a set for model development (`edx`, ~90%) using a modified version of the R code provided in the

course materials. For computational efficiency when re-running the analysis, the downloaded dataset and prepared data objects are saved in `./data/` and re-used if already existing.

Data exploration and feature engineering

For developing a well-performing machine learning model, it is essential to explore the dataset before deciding on a modeling approach. Therefore, in this section the structure and properties of the `edx` set are examined and potentially relevant input features are prepared.

Overall structure

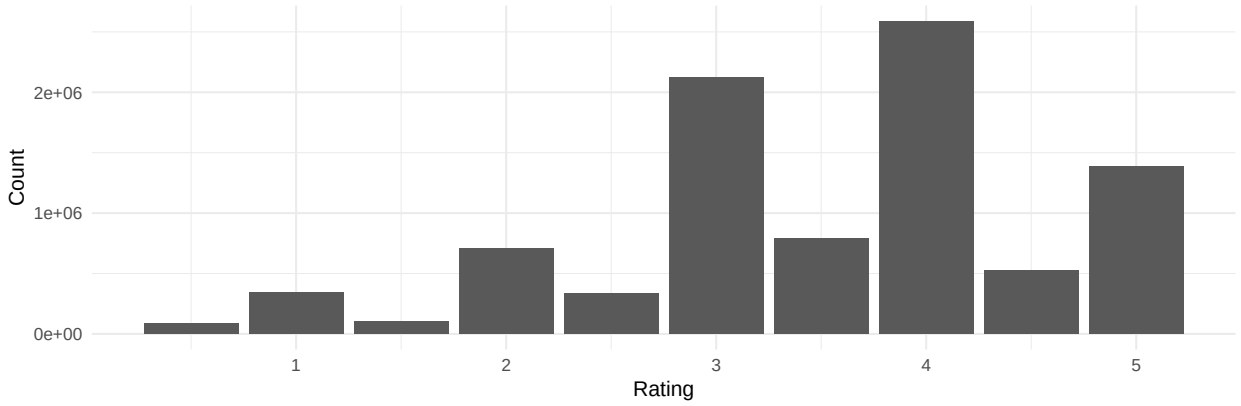
```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 42...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)",...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|...
```

The training dataset contains 6 columns (userId, movieId, rating, timestamp, title, genres) and 9000055 rows. It comprises 69878 user IDs and 10677 movie IDs/10676 titles as well as 20 genre categories and 797 combinations of one or more genres. Each title also includes the publication year in parentheses.

Since the datetime of ratings is only given as integer timestamp and the publication year is part of the title, inferred variables are added as potential additional features.

```
# add publication date extracted from title and rating date, year and weekday
edx <- edx %>%
  mutate(publication_year = str_extract(title, "(?<=\\()[0-9]{4}(?=\\)$)"),
         rating_date = as_datetime(timestamp),
         rating_year = year(rating_date),
         rating_weekday = wday(rating_date, label = T, week_start = 1))
```

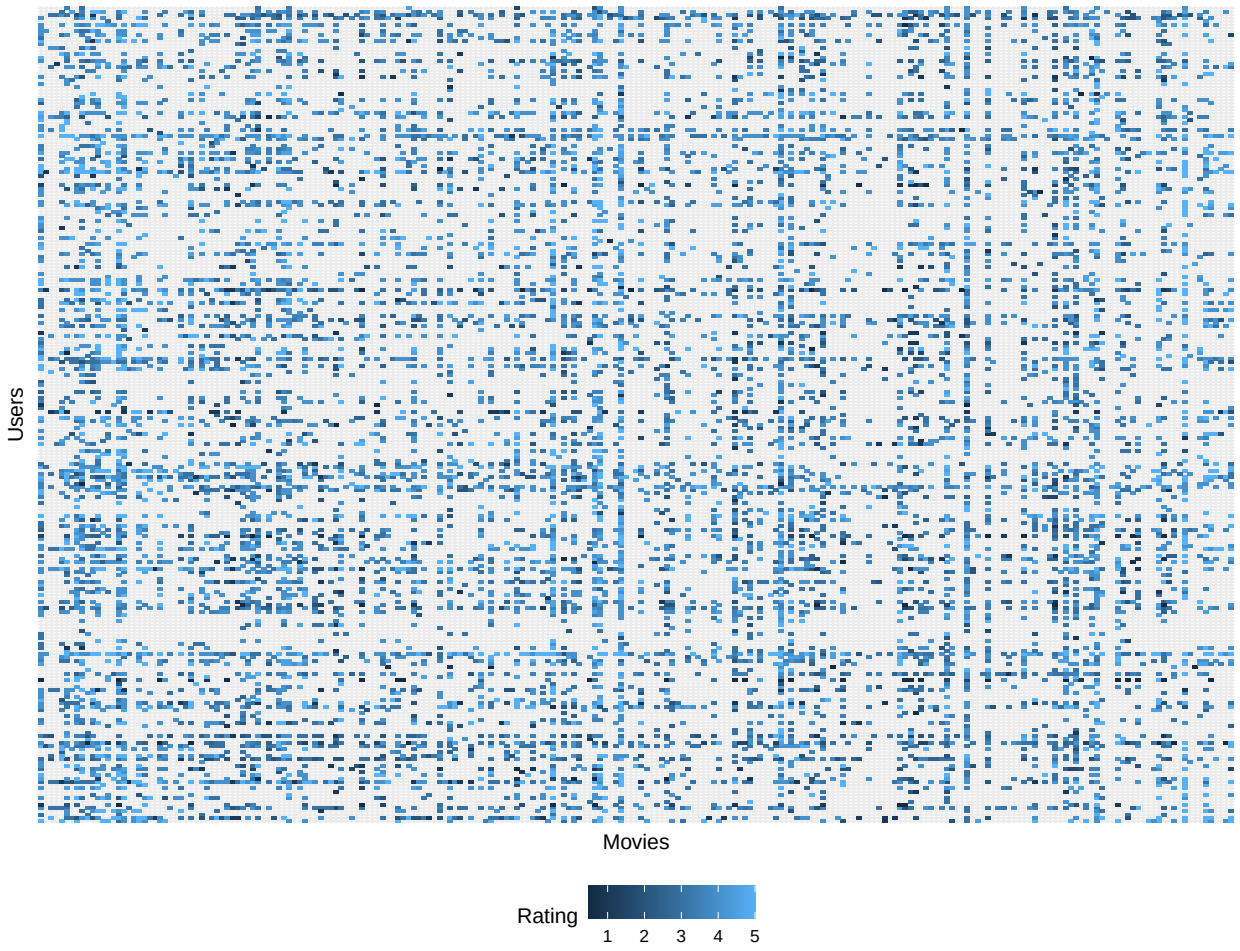
Ratings



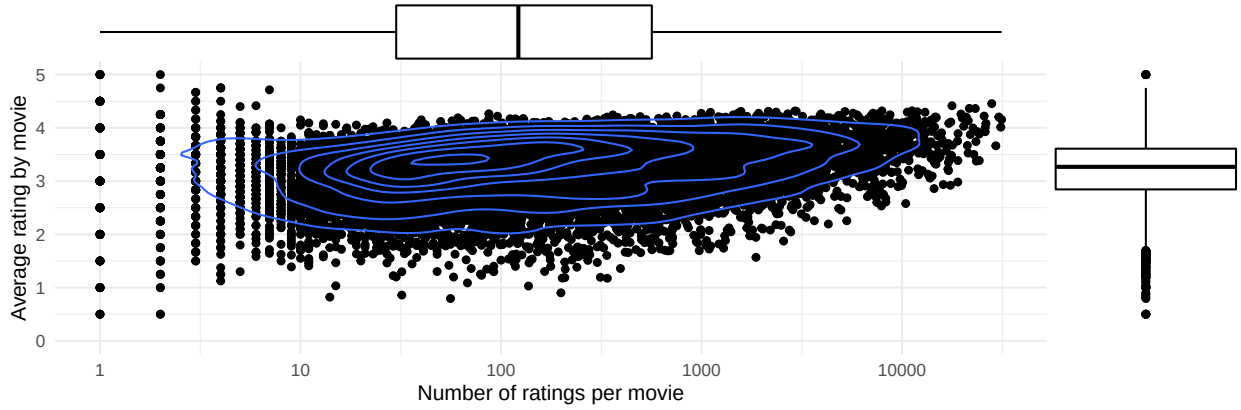
Movie ratings are recorded within a 5-star system that allows for half-star and full-star ratings. 0-star ratings are not allowed. In general, full-star ratings are more common than half-star ratings, although the overall distribution is similar with 4 stars as most frequent full-star rating and 3.5 stars as most frequent half-star rating.

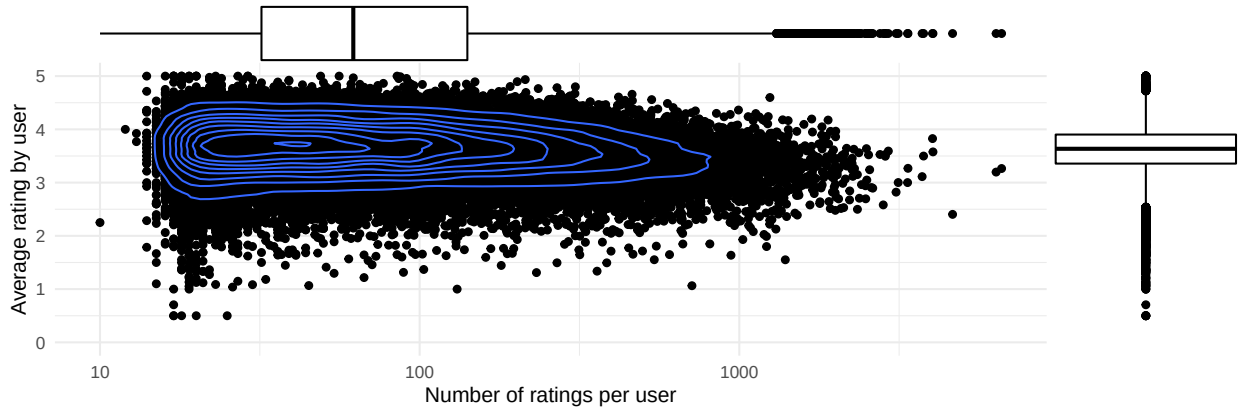
Movies and users

Trends regarding movie-level and user-level data distribution are visualized in the following plots. To evaluate sparsity within the user ID x movie ID rating matrix, a heatmap of ratings for a random subsample of 250 users and 250 movies is shown.



A wide range in the number of ratings per movie and per user is present with some users having rated up to 61.96 % of all available movies and some movies having been rated by up to 44.88 % of all available users.

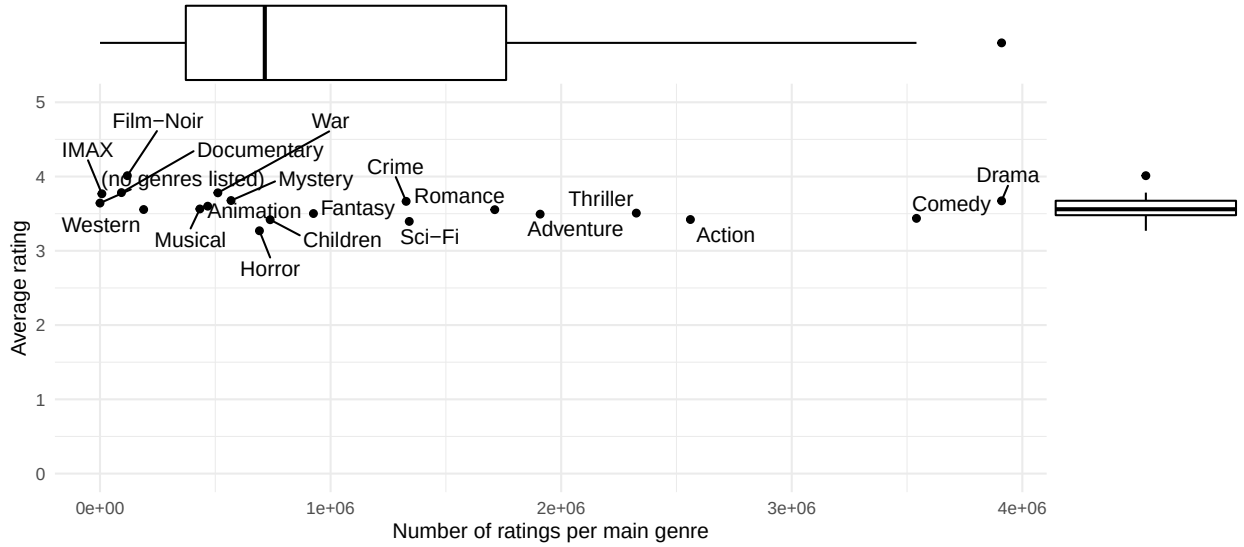




On average, movies with a higher number of ratings, i.e. popular movies, tend to have a higher rating than movies with a low number of ratings, for which a broad range of average ratings is observed. In contrast, on a per-user level the relation between the average rating and the number of ratings per user is not as strong, even though users with a very high number of rated movies tend to have a below-average mean rating. Unsurprisingly, the variability of rating averages per movie or per user decreases with the number of ratings.

Genres

The `genres` column in the dataset contains a set of one or more main genres per movie. In the following plots, the properties of both individual main genres as well as genre combinations are examined.



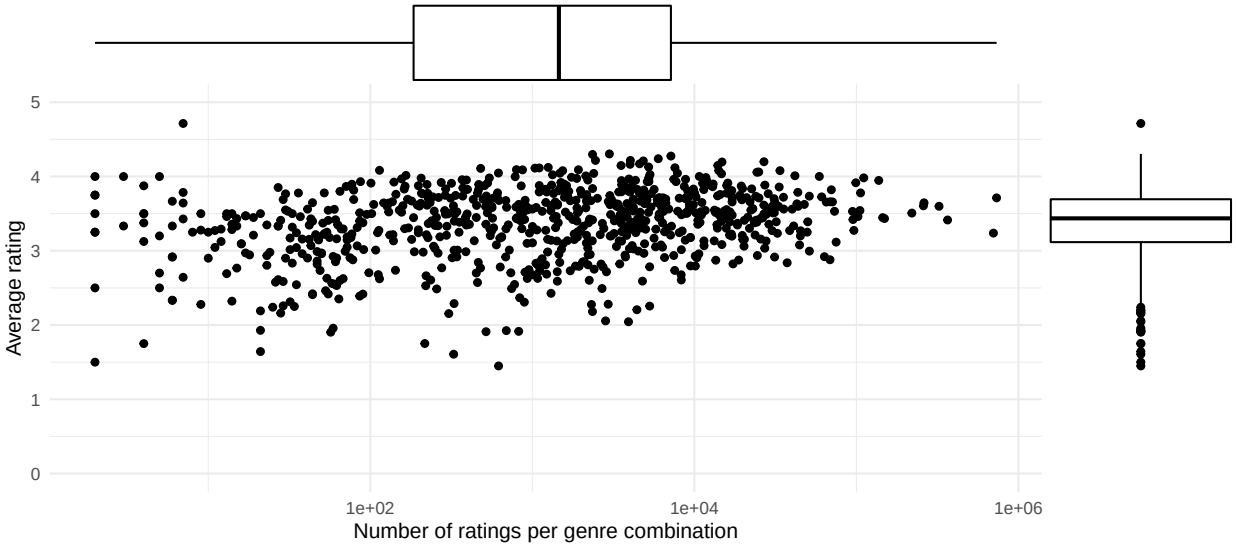


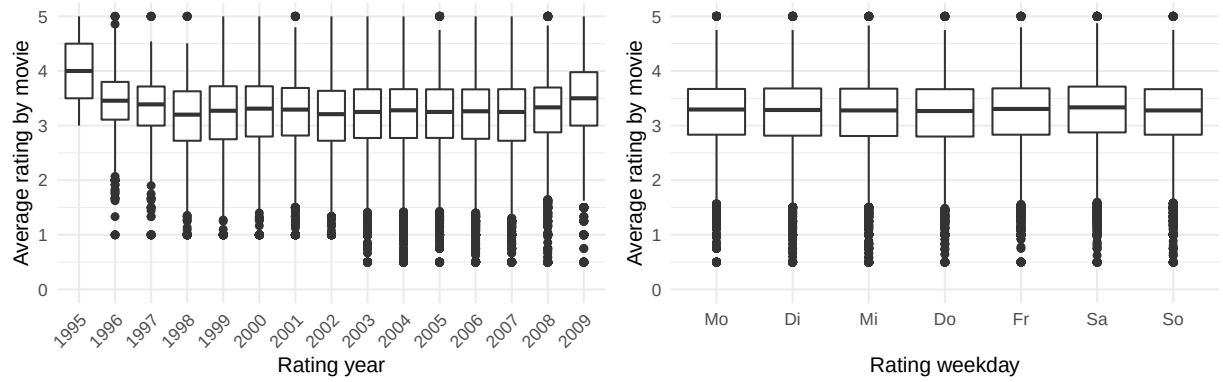
Table 1: Top 5 most frequent genre combinations

Genre combination	Number of ratings	Mean rating
Drama	733296	3.712364
Comedy	700889	3.237858
Comedy|Romance	365468	3.414486
Comedy|Drama	323637	3.598961
Comedy|Drama|Romance	261425	3.645824

For the 20 main genres, only a very small impact on average movie ratings is visible, with rating means per genre ranging from 3.27 for Horror to 4.01 for Film-Noir. In contrast, a moderate effect of genre combinations on ratings is visible, with group means ranging from 1.45 for Documentary|Horror to 4.71 for Animation|IMAX|Sci-Fi. Again, with lower numbers of ratings per genre combination, the variability in average ratings increases.

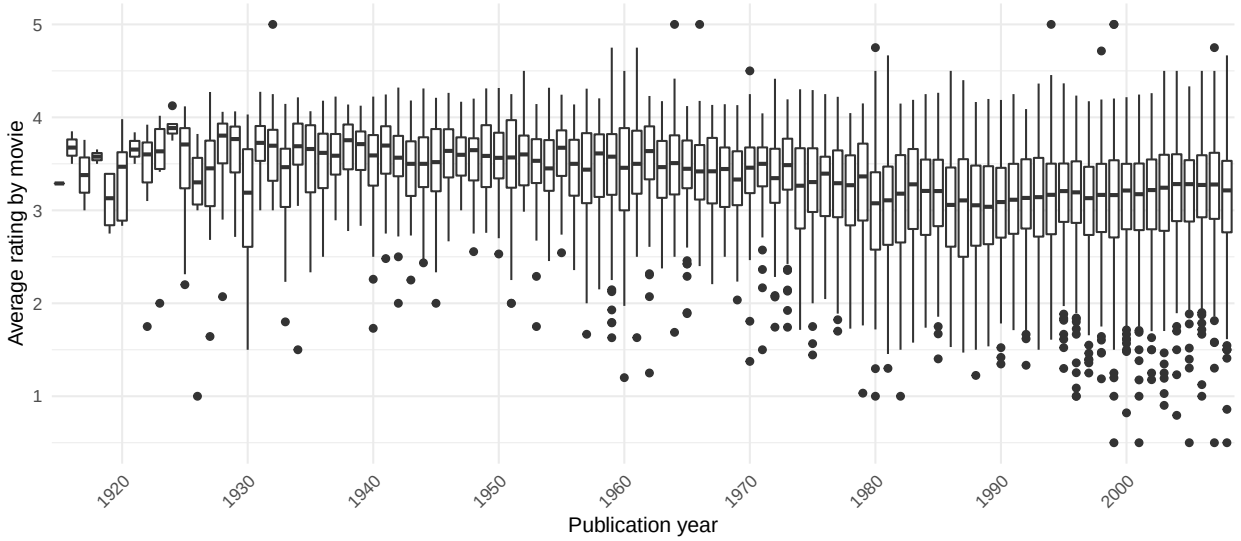
Publication year and rating datetime

Time-related trends are visualized in the following plots.



The rating year seems to have only a small impact on average movie ratings, especially when considering

rating years with very few data points as rather noisy outliers. The effect by rating weekday seems to be even smaller, with the mean of average ratings per movie ranging from 3.18 to 3.24 across all weekdays.



The effect of publication year on average movie ratings ranges from 3 to 3.9 as rating means per publication year.

Model development

For both model development and evaluation, the root mean squared error (RMSE) with predicted ratings $\hat{Y}$, true ratings Y and number of ratings N is used as loss function:

$$RMSE = \sqrt{\frac{1}{N}\sum \hat{Y} - Y}$$

```
# function to calculate root mean squared error
RMSE <- function(Y_hat, Y) sqrt(mean((Y_hat - Y)^2))
```

From the preceding data exploration it can be assumed that both movie-specific and user-specific effects are important for modeling movie ratings. Further, the specified combination of movie genres seems to provide additional information on movie ratings and are thus included in the model as well. Rating year, rating weekday and publication year, however, seem to have a rather small impact on ratings and are therefore not included in the model.

The resulting model for movie ratings Y can be written as follows with the overall average rating μ as baseline and level-specific feature effects b for movie m, user u and movie genre g as well as an error term ϵ:

$$Y = \mu + b_m + b_u + b_g + \epsilon$$

As baseline, μ is estimated as the overall average rating.

```
# estimate mean rating
mu_hat <- mean(edx$rating)
```

In a stepwise approach which reflects the assumed relevance of predictor variables, the individual effects b for available factor levels are estimated according to the following sequence of equations:

$$\hat{b}_m = \frac{1}{n_m}\sum Y_m - \hat{\mu}$$

$$\hat{b}_u = \frac{1}{n_u}\sum Y_u - \hat{\mu} - \hat{b}_m$$

$$\hat{b}_g = \frac{1}{n_g}\sum Y_g - \hat{\mu} - \hat{b}_m - \hat{b}_u$$

```
# estimate model coefficients for movie, user and genre effects
movie_basic <- edx %>%
  group_by(movieId) %>% summarize(b_m = mean(rating - mu_hat))

user_basic <- edx %>% left_join(movie_basic) %>%
  group_by(userId) %>% summarize(b_u = mean(rating - mu_hat - b_m))

genre_basic <- edx %>% left_join(movie_basic) %>% left_join(user_basic) %>%
  group_by(genres) %>% summarize(b_g = mean(rating - mu_hat - b_m - b_u))
```

Since for some movies, users and genres only very few ratings are available, regularization of coefficients is additionally used to improve model performance. Here, the following function will be minimized to obtain the best value for the penalty parameter λ:

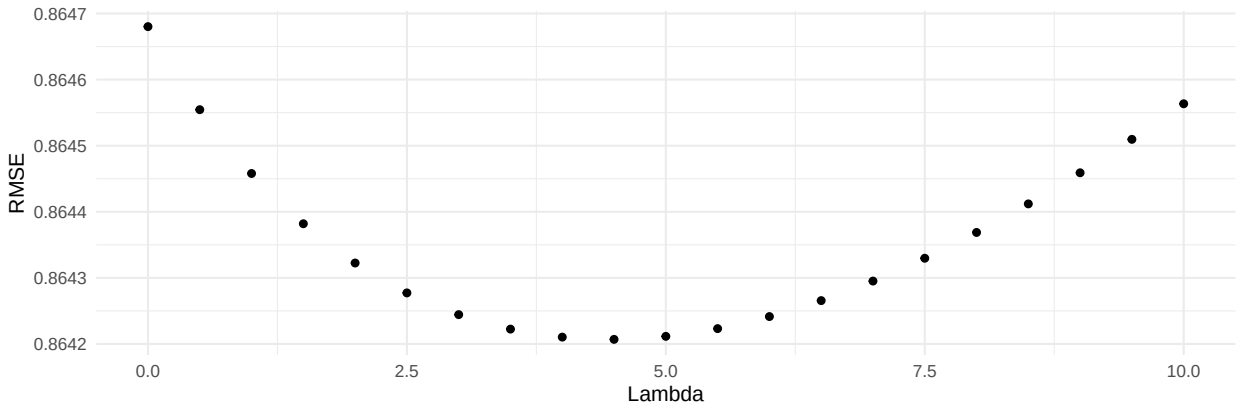
$$\frac{1}{N}\sum(Y - \mu - b_m - b_u - b_g)^2 + \lambda(\sum b_m^2 + \sum b_u^2 + \sum b_g^2)$$

As before in the basic estimation of coefficients, the stepwise approach is employed. Since the penalty λ is a tunable parameter, the `edx` set is split into a training set `train` and a testing set `test` to chose the optimal λ that minimizes the RMSE in the testing set. Using the `train` set to calculate regularized coefficients at different values for λ and the `test` set to calculate the corresponding RMSE, the optimal value for λ that minimizes the RMSE on the test set is then selected.

```
# split edx set into training and testing set for model optimization
if (rver < 360) {set.seed(1)} else {set.seed(1, sample.kind = "Rounding")}
train_index <- createDataPartition(edx$rating, p = 0.9, list = F)

train <- edx[train_index,]
tmp <- edx[-train_index,]

# make sure userId and movieId in test set are also in train set
test <- semi_join(tmp, train, by = "movieId") %>% semi_join(train, by = "userId")

# add rows removed from test set back into train set
train <- rbind(train, anti_join(tmp, test))
```

```
# calculate RSME at different lambdas (regularizing movie, user and genre effects)
lambdas <- seq(0, 10, 0.5)
rmse_reg <- sapply(lambdas, function(lambda) {

  # regularize movie effects
  movie_reg <- train %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu_hat)/(lambda + n()))

  # regularize user effects
  user_reg <- train %>%
```

```
    left_join(movie_reg, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu_hat - b_m)/(lambda + n()))

  # regularize genre effects
  genre_reg <- train %>%
    left_join(movie_reg, by = "movieId") %>%
    left_join(user_reg, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu_hat - b_m - b_u)/(lambda + n()))

  # estimate RSME for test set
  test %>%
    left_join(movie_reg, by = "movieId") %>%
    left_join(user_reg, by = "userId") %>%
    left_join(genre_reg, by = "genres") %>%
    mutate(rating_hat = mu_hat + b_m + b_u + b_g) %>%
    summarize(rmse = RMSE(rating, rating_hat)) %>%
    pull(rmse)
})
```



As the plot demonstrates, the minimum RMSE is obtained at $\lambda = 4.5$. With this optimal value, the final regularized model coefficients are calculated.

```
# calculate regularized coefficients with selected lambda for the entire edx set
movie_reg <- edx %>%
  group_by(movieId) %>%
  summarize(b_m = sum(rating - mu_hat)/(lambda + n()))
user_reg <- edx %>%
  left_join(movie_reg, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_hat - b_m)/(lambda + n()))
genre_reg <- edx %>%
  left_join(movie_reg, by = "movieId") %>%
  left_join(user_reg, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu_hat - b_m - b_u)/(lambda + n()))
```

Results

After fitting the basic and regularized models as described before in the methods section, performance is evaluated by calculating the RMSE on the validation set. For comparison purposes, the RMSE obtained when using the overall average rating as naive prediction is also included.

```
# estimate RMSE for naive model (overall mean)
rmse_naive_mean <- RMSE(validation$rating, mu_hat)

# estimate RMSE for basic model
rmse_basic <- validation %>%
  left_join(movie_basic) %>%
  left_join(user_basic) %>%
  left_join(genre_basic) %>%
  mutate(rating_hat = mu_hat + b_m + b_u + b_g) %>%
  summarize(rmse = RMSE(rating, rating_hat)) %>%
  pull(rmse)

# estimate RMSE for regularized model
rmse_regularized <- validation %>%
  left_join(movie_reg, by = "movieId") %>%
  left_join(user_reg, by = "userId") %>%
  mutate(rating_hat = mu_hat + b_m + b_u) %>%
  summarize(rmse = RMSE(rating, rating_hat)) %>%
  pull(rmse)
```

Table 2: RSME estimates for different modeling approaches

Model	RMSE
naive mean	1.0612018
simple additive model	0.8649469
regularized additive model	0.8648242

Compared to the naive model, in which all ratings are predicted as the overall average, with an RMSE of 1.0612, the simple additive model, in which movie effects, user effects and genre effects are considered, yields a considerably lower RMSE of 0.86495. The regularization of coefficients leads to a further improvement of model performance, however, with a RMSE of 0.86482 this improvement is rather minor.

Conclusion

In this project, a movie recommendation system was implemented using a regularized additive linear regression model with movie effects, user effects and genre effects as predictors. A final RMSE of 0.86482 on the validation set was obtained, which satisfies the requirements specified in the course instructions.

Within the employed model architecture, the inclusion of additional features like publication year or rating year may have yielded an even lower RMSE, however, since the improvement was expected to be small, a more simplistic model was favoured in this context.

Considering that this project is part of the HarvardX's Data Science Professional Certificate, the selection of a modeling approach was to some extend based on the lessons learned in the preceding HarvardX's Data Science Machine Learning course. Thus, further improvements in model performance may be achieved by employing more advanced approaches not extensively covered in the course such as matrix factorization. This approach makes use of patterns within the dataset but is also more expensive in model development.

Session info

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 16299)
##
## Matrix products: default
##
## Random number generation:
##  RNG:     Mersenne-Twister
##  Normal:  Inversion
##  Sample:  Rounding
##
## locale:
## [1] LC_COLLATE=German_Germany.1252  LC_CTYPE=German_Germany.1252
## [3] LC_MONETARY=German_Germany.1252 LC_NUMERIC=C
## [5] LC_TIME=German_Germany.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] ggExtra_0.9       ggrepel_0.8.1     scales_1.0.0      patchwork_1.0.0
##  [5] rpart_4.1-15      glmnet_3.0-2      Matrix_1.2-17     caret_6.0-86
##  [9] lattice_0.20-38   lubridate_1.7.4   data.table_1.12.6 forcats_0.4.0
## [13] stringr_1.4.0     dplyr_1.0.0       purrr_0.3.4       readr_1.3.1
## [17] tidyr_1.1.0       tibble_3.0.3      ggplot2_3.3.2     tidyverse_1.3.0
## [21] here_0.1
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-140         fs_1.3.1             httr_1.4.1
##  [4] rprojroot_1.3-2      tools_3.6.1          backports_1.1.5
##  [7] utf8_1.1.4           R6_2.4.0             DBI_1.1.0
## [10] colorspace_1.4-1     nnet_7.3-12          withr_2.1.2
## [13] tidyselect_1.1.0     compiler_3.6.1       cli_2.0.2
## [16] rvest_0.3.5          xml2_1.2.2           isoband_0.2.2
## [19] labeling_0.3         digest_0.6.22        rmarkdown_2.0
## [22] pkgconfig_2.0.3      htmltools_0.4.0      highr_0.8
## [25] fastmap_1.0.1        dbplyr_1.4.2         rlang_0.4.7
## [28] readxl_1.3.1         rstudioapi_0.11      shiny_1.4.0
## [31] shape_1.4.4          generics_0.0.2       jsonlite_1.6
## [34] ModelMetrics_1.2.2.2 magrittr_1.5         Rcpp_1.0.2
## [37] munsell_0.5.0        fansi_0.4.0          lifecycle_0.2.0
## [40] stringi_1.4.3        pROC_1.16.2          yaml_2.2.0
## [43] MASS_7.3-51.4        plyr_1.8.4           recipes_0.1.13
## [46] grid_3.6.1           promises_1.1.0       crayon_1.3.4
## [49] miniUI_0.1.1.1       haven_2.2.0          splines_3.6.1
## [52] hms_0.5.2            knitr_1.26           pillar_1.4.6
## [55] reshape2_1.4.3       codetools_0.2-16     stats4_3.6.1
## [58] reprex_0.3.0         glue_1.4.1           evaluate_0.14
## [61] modelr_0.1.5         vctrs_0.3.2          httpuv_1.5.2
## [64] foreach_1.4.7        cellranger_1.1.0     gtable_0.3.0
## [67] assertthat_0.2.1     xfun_0.12            gower_0.2.1
## [70] mime_0.7             prodlim_2019.11.13   xtable_1.8-4
```

```
## [73] broom_0.7.0          later_1.0.0          class_7.3-15
## [76] survival_2.44-1.1    timeDate_3043.102    iterators_1.0.12
## [79] lava_1.6.7           ellipsis_0.3.0       ipred_0.9-9
```