Building a movie recommendation system using the MovieLens (10MM) dataset

HarvardX Data Science Professional Certificate

12 June, 2024

Summary: In this project, I develop a movie recommendation system using the MovieLens dataset. The approach used utilizes biases associated with movies, users, genres, and years to address the challenge of excessive predictors in regression analysis, aiming to strike a balance between complexity and predictive accuracy through regularization. However, signs of underfitting are observed due to the model's simplicity. Exploring alternative methodologies to tackle computational constraints could enhance the model's effectiveness, potentially involving distributed computing frameworks or algorithmic optimizations. Despite limitations, this project showcases an innovative approach to predictive modeling, integrating biases and regularization techniques to overcome regression analysis challenges. Future enhancements may focus on refining computational efficiency and feature engineering methodologies to bolster the model's performance.

I. Introduction

Objective

In this project I develop a movie recommendation system using the ten-million-row version of the MovieLens dataset. This dataset was issued and is actively maintained by GroupLens Research, a research group at the University of Minnesota.

Movie recommendation systems analyze movie preferences of users, typically revealed through their ratings. By examining these ratings, the system identifies patterns and similarities among users' tastes and preferences. Leveraging this information, it then identifies similarities between the preferences of different users for movies. This process enables the system to make recommendations for movies that align with the interests of a potential viewer.

The ultimate objective of this project is to develop a recommendation system with a loss function, specifically defined as the root mean squared error (RMSE), where error is defined as the difference between the actual and the estimated rating. More specifically, RMSE can be expressed as follows:

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

where n is the number of ratings, y_i is the actual rating, and $\hat{y}_i$ is the predicted value.

The movielens datset

The **movielens** version used in this project contains 10,000,054 ratings from 69,878 unique individuals, rating 10,677 distinct movies. This version, named edx, was partitioned to create a holdout set containing 10 percent of the data. As will be explained later, the remaining 90% was further partitioned for validation purposes.

Table 1 below shows details of the variables in the dataset, including brief explanations and their respective classes.

Table 1: Variables in the movielens dataset

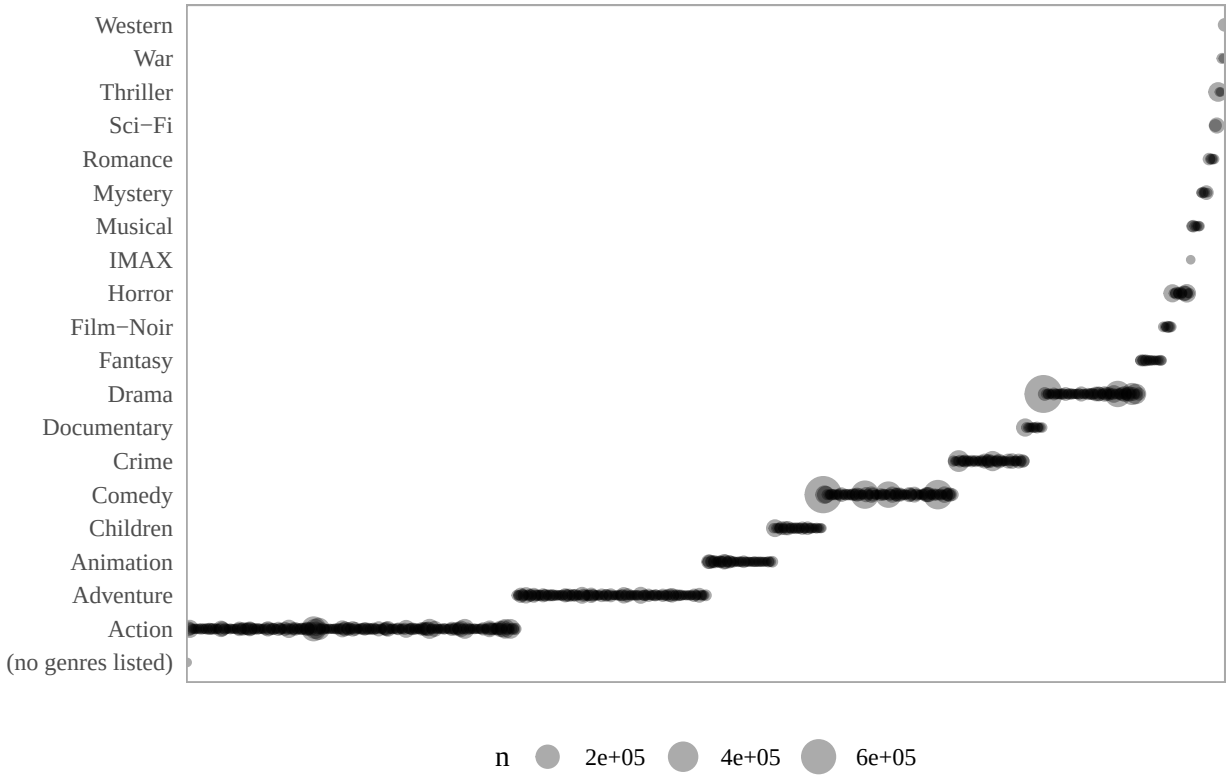
Name	Class	Brief explanation	Used as
movieId	integer	Unique identifier assigned to each movie	Feature
rating	numeric	Valuation given by users to movies	Target
userId	integer	Unique identifier assigned to each user	Feature
title	character	Title of movie, including its year of release in parentheses	Feature
genres	character	Categories that group films by some measure of similarity	Feature
timestamp	integer	Seconds since Jan. 1, 1970 to the moment of review	Feature

In Table 1:

- *rating* represents the subjective valuation provided by users to each movie they rate. This feature, ranging from 0.5 to 5 in 0.5 increments, is the target variable, that is, the variable that we want to predict in this project.
- *usedId* is a unique identifier for each of the users providing their movie ratings.
- *movieId* and *title* refer to individual movies. There is a one-to-one relationship between *movieId*, which is a unique identifier for each movie, and *title*, which refers to the title of movies, including their year of release in parentheses.[1].
- *genres* categorizes films based on shared themes, styles, and narrative elements. In its original categorization, films are sorted into either single genres or combinations of genres, some of which may overlap. For example, Movie A might belong to a single category like "Comedy," while Movie B could be categorized as "Comedy, Drama, and Romantic." To simplify this categorization, we introduce a transformed feature called *single_genre*. This feature retains either the single genre listed in the original classification or, in cases of multiple genres, the first one listed, assuming it best represents the movie. With this simplified categorization, the number of genres reduces from 797 to 20, thus facilitating simpler graphical representations (see Figure 1 below). However, it is important to note that in modeling the target variable, we retain the original classification, *genres*, as it allows for a finer level of classification and, therefore, more variability and, presumably, less bias.

[1]While *movieId* typically serves as a unique identifier, there is a case where two different identifiers are assigned to a single *title*. Unfortunately, with the available information, it is not possible to determine whether there are two titles and one is missing or there is a single title that was assigned two different identifiers. However, considering the size of our training data and our chosen modeling approach (please refer to Section II for our modeling approach), this ambiguity is not expected to introduce any systematic errors in our results

Figure 1: Mapping of 797 movie genres into 20 broad categories



- *timestamp* is a numerical value representing the specific moment in time when a rating was submitted, measured in seconds elapsed since the start of January 1, 1970. [2] This standardized time format allows for precise recording and comparison of when ratings were provided, aiding in a temporal dimension to our analysis.

[2] Also known as the Unix epoch.

II. Methods

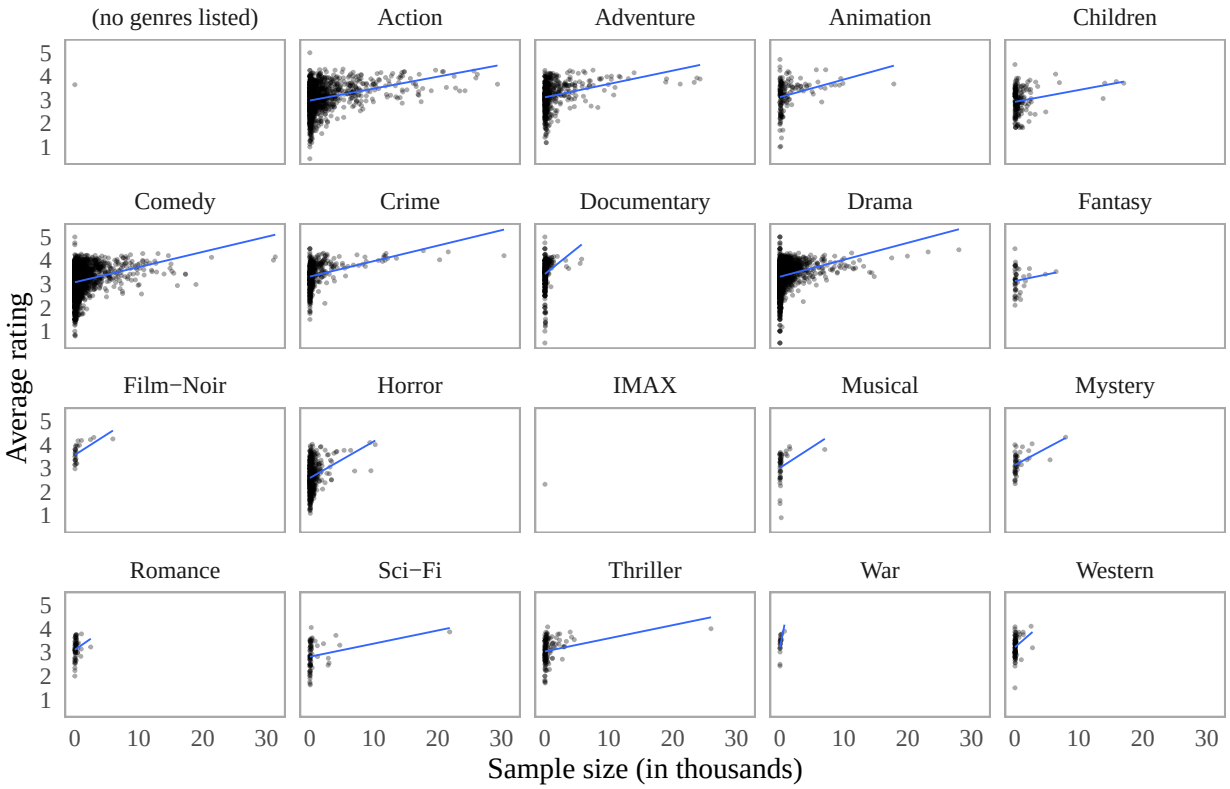
This section provides an overview of the main characteristics of the target variable, *rating*, and the features utilized for its prediction. Our data exploration primarily involves visualizing univariate and bivariate relationships between the outcome and features. It is worth noting that not all insights presented in this section will be incorporated into our modeling approach. The main reason being that although many of these findings could have been integrated using regression analysis, regression was not a feasible option.

Data exploration and visualization

- Average movie ratings and sample size

In Figure 2 below, each data point represents a combination of average rating and number of ratings for each individual movie. The blue line indicates the expected rating for different levels of review counts. As the number of reviews increases along the horizontal axis, the average rating rises and, as expected, its dispersion becomes narrower. [3]

Figure 2: Average rating vs. number of ratings



- Count of ratings by user

Table 2 below presents summary statistics of the distribution of the total number of ratings by individual users. This distribution is rightly skewed (since the mean is about twice the median), and ranges from 10 to an astonishing 6,616.

[3]The noted inverse relationship between average rating and dispersion is a statistical regularity that arises because, in our context, the sampling distribution of the average rating for a particular movie is inversely related to the square root of the number of reviews it receives. For a clear and engaging explanation of this regularity see Wainer (2007).

Table 2: Review count per user

Statistic	Value
Minimum	10
Q1	32
Median	62
Mean	129
Q3	141
Maximum	6616

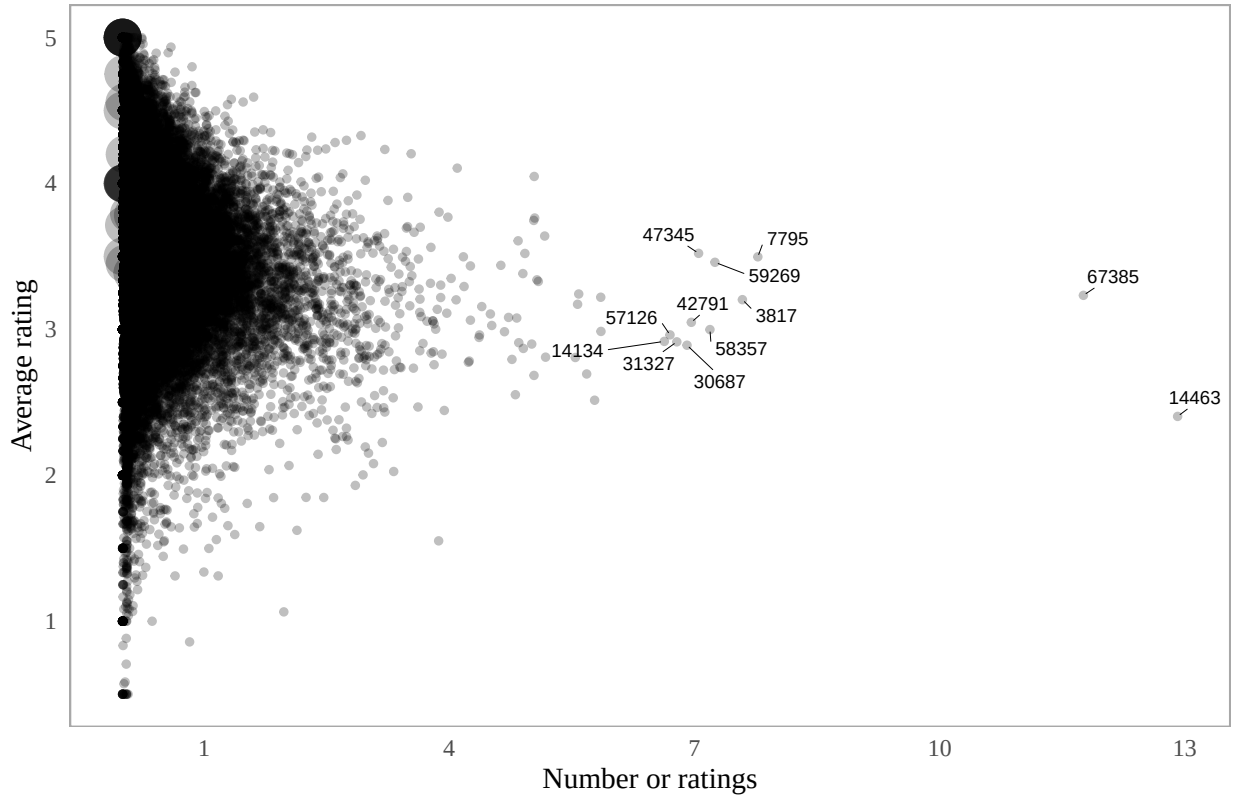
Table 3 presents cumulative values of reported number of ratings by user spanning from 1995 to 2009, which can be challenging to interpret. To provide a clearer perspective on these numbers, Figure 3 below illustrates the relationship between the number of reviews per day and average ratings.

Table 3: Users with atypical (excessively high) daily rating counts

userId	year_reviewed	Yearly	Daily
14463	2002	4648	13
67385	2007	4233	12
7795	2002	2799	8
3817	2006	2731	8
59269	2001	2610	7
58357	2007	2588	7
47345	2007	2538	7
42791	2005	2506	7
30687	2006	2487	7
31327	2006	2442	7
57126	2007	2412	7
14134	2005	2387	7

In this figure, all dots corresponding to users who reported reviewing approximately at least 7 movies per day are annotated by their *userId*. For example, the two rightmost values in this figure indicate that these users reviewed 10 and 13 movies each day on average, respectively. Given that the average duration of movies falls between 90 and 120 minutes, these users may not necessarily have spent between 900 minutes (or 15 hours) and 1200 minutes (20 hours) per day watching movies. This is because *…Ratings in MovieLens can occur at any time, possibly many years after watching a movie. Users often enter a large number of ratings in a single session, either to fill in their rating history for personal satisfaction or in hopes of receiving more personalized recommendations.* (Maxwell and Constant, 2015, p. 15). In any case, it should be clear that these extreme reports put into question the reliability of their corresponding ratings.

Figure 3: Daily movie rating count



- Users giving identical ratings regardless of movie

Similarly, 36 users consistently gave the same ratings to every movie they reviewed. This type of review, void of critical assessment, overlooks the uniqueness and individual characteristics of each film. Consequently, it compromises the validity of these ratings. Table 4 below shows these cases.

Table 4: Users giving identical ratings regardless of movie

userId	Rating	Movies reviewed
29110	3.0	185
24176	1.0	131
52749	5.0	85
68379	5.0	56
18965	5.0	49
33017	4.0	43
49438	3.0	30
13027	5.0	29
15575	5.0	29
54009	5.0	27
48146	0.5	25
31598	4.0	23
35184	5.0	23
4771	3.0	21
13524	5.0	20
27831	5.0	20
42649	5.0	20
62815	0.5	20
1	5.0	19
3457	1.0	19
65873	5.0	19
11884	5.0	18
22045	5.0	18
49209	2.5	18
63381	0.5	18
7984	5.0	17
13496	0.5	17
13513	5.0	17
22964	4.0	17
24490	1.0	17
26308	5.0	17
30519	5.0	17
49862	0.5	17
58344	4.0	17
71422	5.0	16
52674	5.0	14

- Movies

There are 10677 movies in the edx dataset but only 10676 distinct titles. This is because the movie *War of the Worlds (2005)* has two different Ids.

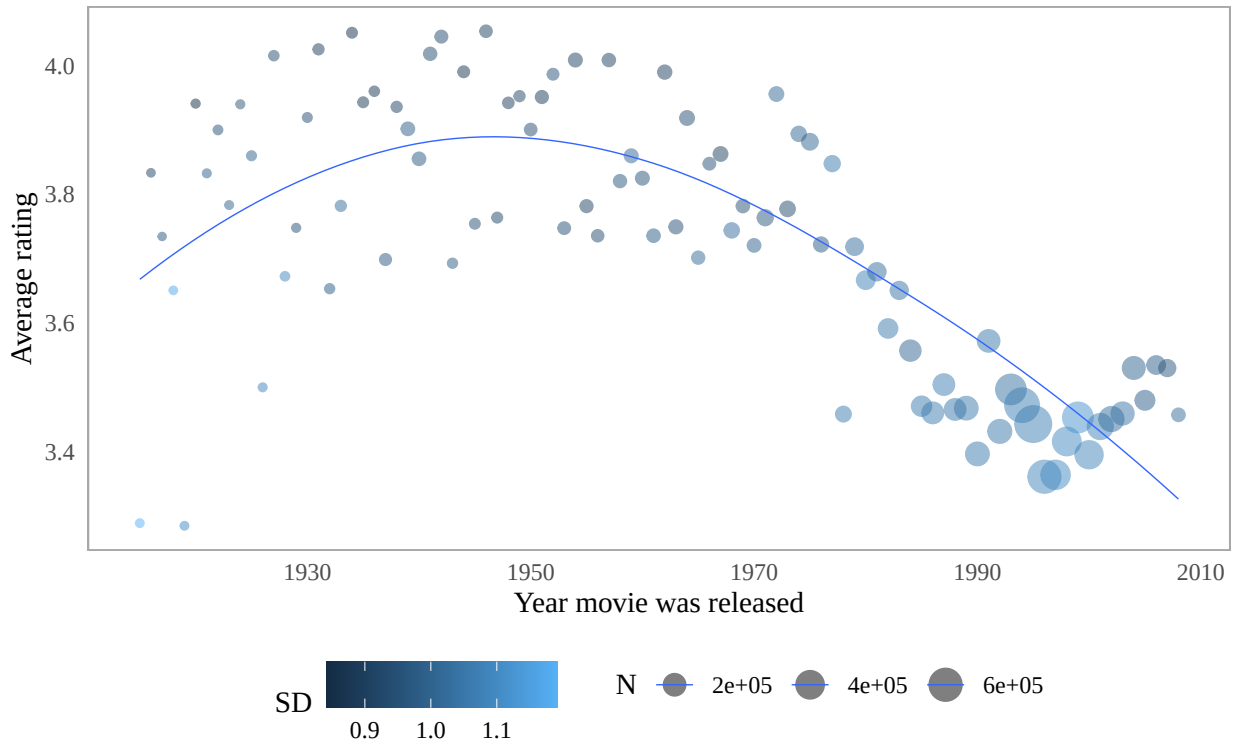
Table 5: War of the Worlds (2005)

movieId	title	n
34048	War of the Worlds (2005)	2460
64997	War of the Worlds (2005)	28

- Average rating by released year

Figure 4 below shows the average rating of movies over time. The dot sizes are proportional to the number of ratings used to estimate these averages. The color represents variability, measured by the standard deviation of ratings. This figure suggests that since the 1970s, average ratings have been lower, though they are based on a significantly larger number of ratings.

Figure 4: Decline in average movie ratings despite increased ratings volume since the 1970s



Modeling approach

The prediction of numerical outcomes, such as average movie ratings, is commonly achieved through regression analysis. However, due to the large number of categories in certain features, notably the *genres* category, estimating a regression model with the available computational resources is practically unfeasible.

Under these circumstances, traditional regression models falter due to the exponential increase in computational complexity, rendering them impractical for analysis. To surmount this issue, I basically follow the approach presented in Irizarry (2020, Chapter 33.7).

Let's define the simplest representation of movie ratings as:

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

where,

- $Y_{u,i}$, denote the observed rating for user u and movie i;
- μ, denote the overall average rating across all users and items; and
- $\epsilon_{u,i}$, is an error term associated with the rating for user u and item i. This term captures the discrepancy between the observed rating and the predicted rating from the model.

Adding movie effects

In what follows, for simplicity, I am ignoring the error term, $\epsilon_{u,i}$.

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

One of the pivotal components of the model is the movie-specific bias $b_{1,i}$, which encapsulates the inherent characteristics of each movie in the dataset. By calculating this bias through a weighted aggregation of the deviations between actual ratings $Y_{u,i}$ and the global average rating (μ) across all users who have rated the movie, the model can estimate the idiosyncrasies of individual movies:

$$b_{1,i} = Y_{u,i} - \mu$$

Adding user effects

$$Y_{u,i} = \mu + b_{1,i} + b_{2,u} + \epsilon_{u,i}$$

Similarly, the user-specific bias $b_{2,u}$ enriches the model's predictive capabilities by capturing the unique rating behaviors exhibited by individual users. This bias delineates the discrepancy between a user's ratings, the global average rating (μ), and the movie-specific bias. In essence, the user specific bias serves as a corrective mechanism, reducing the influence of individual users on the overall rating prediction. It is express as:

$$b_{2,u} = Y_{u,i} - \mu - b_{1,i}$$

Finally, the model extends its predictive prowess by incorporating genre-specific biases ($b_{3,g(i)}$) and year-specific biases (b_y), each playing a crucial role in capturing genre and temporal trends in user preferences, respectively:

Adding genre effects

$$Y_{u,i} = \mu + b_{1,i} + b_{2,u} + b_{3,g(i)} + \epsilon_{u,i}$$

Adding temporal effects

$$Y_{u,i} = \mu + b_{1,i} + b_{2,u} + b_{3,g(i)} + b_y f(t_{u,i}) + \epsilon_{u,i}$$

For the sake of brevity, I will omit the detailed derivation of these effects. However, they can be easily obtained by following the same approach used previously for calculating the earlier effects.

III. Results

The first step in modeling movie ratings involves creating a new training set (see code below) from the edx dataset to avoid data leakage and ensure robust model evaluation. Data leakage occurs when information from the test set inadvertently influences the training process, leading to overly optimistic performance estimates. To mitigate this risk, a new training set is constructed by excluding movies and users that appear in the final holdout and test sets. This ensures that the model is trained only on information that would realistically be available at the time of prediction, enhancing its ability to generalize to unseen data.

```r
# Creating a training set
set.seed(1, sample.kind = 'Rounding')
test_index <-  createDataPartition(y = edx$rating, times = 1, p = 0.1, list = F)
training_set <-  edx[-test_index,]
temp <-  edx[test_index,]
# Make sure userId and movieId in edx_test set are also in training_set
edx_test <-  temp |>
semi_join(training_set, by = 'movieId') |>
semi_join(training_set, by = 'userId') |>
as.data.table()
# Add rows removed from edx_test set back into training_set
removed_new <-  anti_join(temp, edx_test)
training_set <-  rbind(training_set, removed_new) |>
  as.data.table() # for faster computation
rm(removed_new, temp, test_index)
```

This separation of data allows for a more accurate assessment of the model's performance on unseen data. Additionally, by incorporating a validation set for hyperparameter tuning and a separate test set for final evaluation, the integrity of the modeling process is maintained, enabling reliable predictions.

Table 6 below presents the performance of different models in predicting ratings, each model accounting for increasingly complex factors. The first model, relying solely on the global average rating, yields an RMSE of 1.0603622, indicating the error in predictions made using only this simple metric. As the models become more complex, incorporating factors such as movie effects, user effects, genre effects, and year effects, the RMSE values steadily decrease. For instance, including movie effects alone reduces the RMSE to 0.9439729, suggesting a notable improvement in prediction accuracy compared to the simplistic global average approach. Subsequent models incorporating user, genre, and year effects show further reductions in RMSE, indicating enhanced predictive power as more nuanced factors are considered. Overall, this progression highlights the importance of accounting for various factors in accurately predicting ratings, with each additional factor contributing to a more precise model.

Table 6: RMSE

Model	RMSE
Simple, global average	1.0603622
Movie effects	0.9439729
Movie and user effects	0.8658528
Movie, user and genre effects	0.8654490
Movie, user, genre and year effects	0.8654322

However, despite the improvements observed, further enhancements in prediction accuracy can be achieved through the use of regularization techniques. Regularization is a method for preventing overfitting and improving the generalization ability of models by penalizing overly complex parameter estimates. By adding a regularization term to the loss function, models favor simpler solutions, effectively balancing between fitting the training data well and avoiding excessive complexity.

One common regularization technique used in linear models, including collaborative filtering models, is ridge regression, also known as L_2 regularization. Ridge regression adds a penalty term proportional to the square of the magnitude of the coefficients, thereby shrinking their values towards zero without eliminating them entirely. This helps to mitigate the problem of multicollinearity and reduces the risk of overfitting by imposing constraints on the model's parameter estimates.

In the context of collaborative filtering, regularization is particularly valuable in preventing the model from learning overly specific patterns from the training data that may not generalize well to unseen data.

Optimizing the regularization strength, typically controlled by a hyperparameter λ, is crucial for achieving optimal model performance. The choice of λ involves a trade-off between model simplicity and predictive accuracy. A small λ allows for more flexibility in the model but may lead to overfitting, while a large λ imposes stronger regularization, potentially resulting in underfitting. In what follows λ will be tuned through cross-validation or grid search is essential to find the optimal balance and maximize the model's predictive power.

Tuning lambda

The code below conducts cross-validation to tune the regularization parameter λ, which controls the strength of regularization. It defines a function, `RMSEs`, to calculate RMSE using regularization with λ. This function computes biases for movies, users, genres, and years, incorporates these biases into predictions, and returns the RMSE for a given λ.

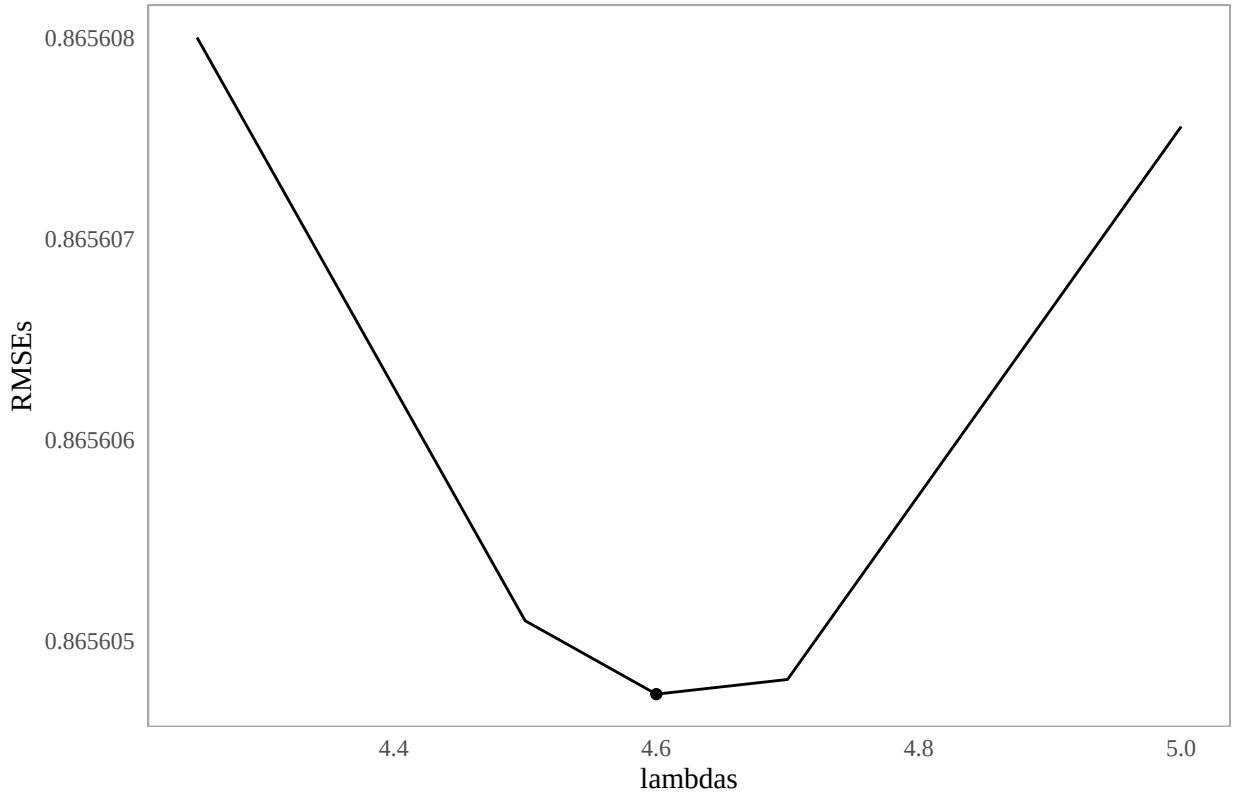
Next, the code sets up cross-validation with 5 folds and specifies a sequence of λ values to evaluate. Within a nested loop, it iterates over each λ value and each fold, extracting training and test data for each fold and calculating the RMSE using the `RMSEs` function. After completing cross-validation, the code calculates the mean RMSE for each λ and identifies the optimal λ value that minimizes the mean RMSE.

```
# Function for tuning lambda
set.seed(1)

RMSEs <- function(train_data, test_data, lambda) {
  mu <- mean(train_data$rating)

# Convert train_data to data.table for faster computations
  train_dt <- as.data.table(train_data)

# Movie bias
  b_i <- train_dt[, .(b_i = sum(rating - mu) / (.N + lambda)),
                  by = movieId]
# User bias
  b_u <- merge(train_dt, b_i[, .(movieId, b_i)],
               by = "movieId", all.x = TRUE)[,
               .(b_u = sum(rating - b_i - mu) / (.N + lambda)), by = userId]
# Gender bias
  b_g <- merge(merge(train_dt, b_i, by = "movieId", all.x = TRUE),
               b_u, by = "userId", all.x = TRUE)[,
               .(b_g = sum(rating - mu - b_i - b_u) / (.N + lambda)),
               by = genres]

  # Year bias (year_released)
  b_y <- merge(merge(merge(train_dt, b_i, by = "movieId", all.x = TRUE),
              b_u, by = "userId", all.x = TRUE),
              b_g, by = "genres", all.x = TRUE)[,
              .(b_y = sum(rating - mu - b_i - b_u - b_g) / (.N + lambda)),
              by = year_released]

# Convert test_data to data.table for faster join
  test_dt <- as.data.table(test_data)

  predicted_ratings <- merge(
    merge(merge(merge(test_dt, b_i, by = 'movieId', all.x = TRUE),
              b_u, by = 'userId', all.x = TRUE),
```

```
              b_g, by = 'genres', all.x = TRUE),
              b_y, by = 'year_released', all.x = TRUE)[,
                pred := mu + b_i + b_u + b_g + b_y][!is.na(pred)]

return(RMSE(predicted_ratings$pred, predicted_ratings$rating))
}

n_folds <- 5 # 2 folds 72.89 secs; 5 folds: 204.327 secs; 10 folds: 435.717; 20 folds: a lot

# Create folds
  folds <- vfold_cv(training_set, v = n_folds)
# Lambdas
lambdas <-  c(4.25, seq(4.5, 4.7, by = .1), 5)
# Matrix to store RMSEs
rmse_matrix <- matrix(NA, nrow = length(lambdas), ncol = n_folds)

cl <- makeCluster(detectCores() - 1)
registerDoParallel(cl)

for (i in seq_along(lambdas)) {
  for (j in seq_along(folds$splits)) {
    train_data <- analysis(folds$splits[[j]])
    test_data  <- assessment(folds$splits[[j]])
    rmse_matrix[i, j] <- RMSEs(train_data, test_data, lambdas[i])
  }
}
stopCluster(cl)
# Mean RMSEs for each lambda
mean_rmse <- rowMeans(rmse_matrix)

# Lambda that minimizes RMSE
optimal_lambda <- lambdas[which.min(mean_rmse)]
```

Since RMSE is a quadratic function, any point where its curve changes direction, known as an inflection point, can be used to identify the minimum RMSE. In Figure 5 below, the combinations of lambdas and RMSE are depicted, with the minimum $\lambda = 4.6$ value indicated by a dot.

Figure 5: RMSE vs. lambdas



Regularization

In the context of movie ratings prediction, L2 regularization serves as a tool to mitigate overfitting and enhance the generalization capability of results. L2 regularization, also known as Ridge regression, introduces a penalty term to the model's cost function, discouraging excessively large coefficients for predictors. This penalty term is proportional to the square of the magnitude of the coefficients, effectively shrinking them towards zero without eliminating them entirely.

The RMSE of regularized model, 0.8648171, indicates a marginal increase in model performace.

Finally, some of the predictions are inadmissible, as they fall outside the possible rating limits. The acceptable range for ratings is between 0.5 and 5, inclusive ($0.5 \leq \text{rating} \leq 5$). Predictions that do not meet this criterion were excluded from the analysis. After removing these extreme ratings, RMSE was recalculated, resulting in an RMSE of 0.8646991.

Table 7 provides a comprehensive overview of the RMSEs values for each method. Additionally, it includes a column (Growth rate) denoting the improvement in percentage terms from one model relative to the previous, offering some insights into the efficacy of each approach relative to others.

The performance of various models in predicting movie ratings shows significant improvements at each stage of complexity. Starting from the simple global average model with an RMSE of 1.0603622, the introduction of movie-specific effects reduces the RMSE by approximately 11 percent, marking a substantial enhancement. Adding user-specific effects further decreases the RMSE to0.8658528, with an 8 percent improvement, indicating the critical role of user preferences in predictions. Incorporating genre effects leads to a marginal improvement with a 0.05 percent reduction in RMSE, while adding year effects shows an almost negligible enhancement. Regularization slightly improves the model performance, reducing the RMSE by 0.07 percent. Finally, constraining ratings within the range of 0.5 to 5.0 results in a very minor improvement of about 0.01

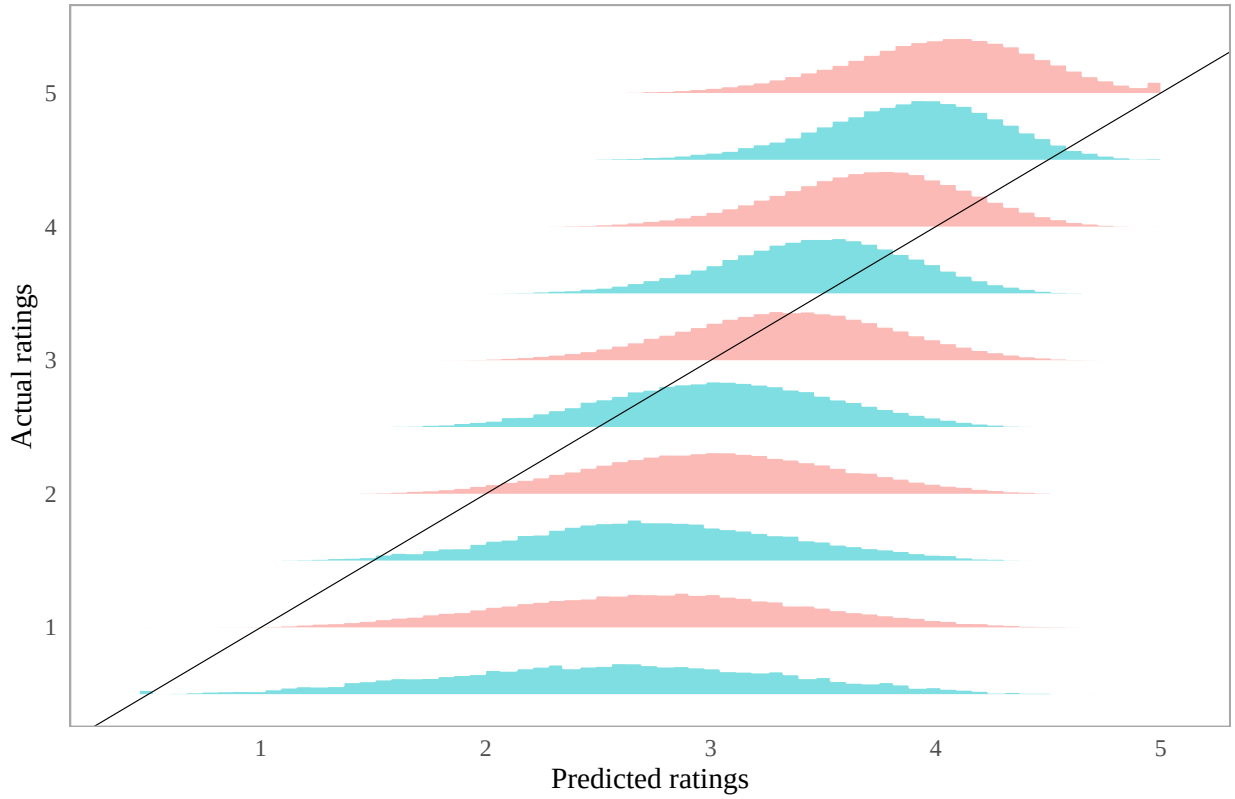
Table 7: RMSE

Model	RMSE	Growth rate
Simple, global average	1.0603622	NA
Movie effects	0.9439729	-10.976
Movie and user effects	0.8658528	-8.276
Movie, user and genre effects	0.8654490	-0.047
Movie, user, genre and year effects	0.8654322	-0.002
Regularized	0.8648171	-0.071
Constrained ($0.5 \leq$ rating ≤ 5.0)	0.8646991	-0.014

percent. Overall, the most notable improvements are observed with the inclusion of movie and user effects, while subsequent refinements provide diminishing returns.

Figure 6 below compares the distribution of predicted ratings along the horizontal axis and the actual ratings along the vertical axis. These distributions are color-coded, with integer ratings depicted in orange and fractional ratings in green. Notably, these distributions show (slightly) reduced variability towards the higher end of the rating scale (5). However, the most significant observation in this figure pertains to the systematic bias apparent in these distributions when compared to the diagonal line, which represents the set of values where predicted and actual ratings coincide (isoline). Notably, the modal value (peak of the distribution) for a rating of 3.5 appears unbiased, as the diagonal line closely aligns with the center of the corresponding distribution. However, the predicted ratings tend to consistently underestimate the actual ratings for values higher than 3.5, as evidenced by the distribution being mostly located to the left of the diagonal line. Conversely, for ratings between 0.5 and 3, most of the distribution tends to be located to the right of the diagonal line, suggesting an overestimation. This indicates that the model might be underfitting and producing biased results due to its simplicity.

Figure 6: Comparing actual and predicted ratings



IV. Conclusion

The model presented uses biases associated with movies, users, genres, and years to address the challenge of excessive predictors in regression analysis, often rendering traditional methods impractical with available computing resources. Through regularization, it seeks to strike a balance between complexity and predictive accuracy, optimizing the regularization parameter (λ) through a tuning process to minimize the loss function (RMSE). However, as observed previously, this approach also displays signs of underfitting, an expected outcome given its simplicity.

Exploring alternative methodologies to tackle computational constraints could enhance the model's effectiveness. This could involve utilizing distributed computing frameworks or implementing algorithmic optimizations tailored for large-scale datasets. Furthermore, integrating feature engineering techniques may enrich the model's predictive capabilities by extracting more informative features from the data.

Despite its limitations, this project presents an interesting approach to predictive modeling, showcasing the integration of biases and regularization techniques to overcome challenges in regression analysis. Future extensions could focus on refining computational efficiency and feature engineering methodologies, ultimately enhancing the model's performance.

References

Irizarry, A. Rafael (2020). *Introduction to Data Science*. CRC Press.

Maxwell Harper, F. and Joseph A. Konstan (2015). *The MovieLens datasets: History and context.* ACM Trans. Interact. Intell. Syst. 5, 4. DOI: http://dx.doi.org/10.1145/2827872

Wainer, Howard (2007). *The Most Dangerous Equation.* American Scientist. Vol. 95, pp. 249 - 256.

Session Info

R version 4.3.3 (2024-02-29)

Platform: aarch64-apple-darwin20 (64-bit)

locale: en_US.UTF-8||en_US.UTF-8||en_US.UTF-8||C||en_US.UTF-8||en_US.UTF-8

attached base packages: *parallel*, *stats*, *graphics*, *grDevices*, *utils*, *datasets*, *methods* and *base*

other attached packages: *ggridges(v.0.5.4)*, *rsample(v.1.2.0)*, *doParallel(v.1.0.17)*, *iterators(v.1.0.14)*, *foreach(v.1.5.2)*, *tictoc(v.1.2)*, *kableExtra(v.1.3.4.9000)*, *ggrepel(v.0.9.3)*, *data.table(v.1.15.0)*, *caret(v.6.0-94)*, *lattice(v.0.22-5)*, *lubridate(v.1.9.3)*, *forcats(v.1.0.0)*, *stringr(v.1.5.1)*, *dplyr(v.1.1.4)*, *purrr(v.1.0.2)*, *readr(v.2.1.5)*, *tidyr(v.1.3.1)*, *tibble(v.3.2.1)*, *ggplot2(v.3.4.4)* and *tidyverse(v.2.0.0)*

loaded via a namespace (and not attached): *tidyselect(v.1.2.1)*, *viridisLite(v.0.4.2)*, *timeDate(v.4022.108)*, *farver(v.2.1.1)*, *fastmap(v.1.1.1)*, *pROC(v.1.18.5)*, *digest(v.0.6.34)*, *rpart(v.4.1.23)*, *timechange(v.0.3.0)*, *lifecycle(v.1.0.4)*, *survival(v.3.5-8)*, *magrittr(v.2.0.3)*, *compiler(v.4.3.3)*, *rlang(v.1.1.3)*, *tools(v.4.3.3)*, *utf8(v.1.2.4)*, *yaml(v.2.3.8)*, *knitr(v.1.45)*, *labeling(v.0.4.3)*, *bit(v.4.0.5)*, *plyr(v.1.8.9)*, *xml2(v.1.3.6)*, *withr(v.3.0.0)*, *nnet(v.7.3-19)*, *grid(v.4.3.3)*, *stats4(v.4.3.3)*, *fansi(v.1.0.6)*, *colorspace(v.2.1-0)*, *future(v.1.32.0)*, *globals(v.0.16.2)*, *scales(v.1.3.0)*, *MASS(v.7.3-60.0.1)*, *cli(v.3.6.2)*, *rmarkdown(v.2.25)*, *crayon(v.1.5.2)*, *generics(v.0.1.3)*, *rstudioapi(v.0.15.0)*, *future.apply(v.1.10.0)*, *httr(v.1.4.7)*, *reshape2(v.1.4.4)*, *tzdb(v.0.4.0)*, *pander(v.0.6.5)*, *splines(v.4.3.3)*, *rvest(v.1.0.3)*, *vctrs(v.0.6.5)*, *webshot(v.0.5.5)*, *hardhat(v.1.3.0)*, *Matrix(v.1.6-5)*, *hms(v.1.1.3)*, *bit64(v.4.0.5)*, *listenv(v.0.9.0)*, *systemfonts(v.1.0.5)*, *gower(v.1.0.1)*, *recipes(v.1.0.8)*, *glue(v.1.7.0)*, *parallelly(v.1.35.0)*, *codetools(v.0.2-19)*, *stringi(v.1.8.4)*, *gtable(v.0.3.5)*, *munsell(v.0.5.0)*, *furrr(v.0.3.1)*, *pillar(v.1.9.0)*, *htmltools(v.0.5.7)*, *ipred(v.0.9-14)*, *lava(v.1.7.2.1)*, *R6(v.2.5.1)*, *vroom(v.1.6.5)*, *evaluate(v.0.23)*, *highr(v.0.11)*, *class(v.7.3-22)*, *Rcpp(v.1.0.12)*, *svglite(v.2.1.3)*, *nlme(v.3.1-164)*, *prodlim(v.2023.08.28)*, *mgcv(v.1.9-1)*, *xfun(v.0.44)*, *pkgconfig(v.2.0.3)* and *ModelMetrics(v.1.2.2.2)*