Content-Based Recommendation System Developed from the MovieLens 10M Dataset

Nicelle Sernadilla Macaspac

June 2023

Introduction

The Big Tech companies use recommendation systems to provide customized suggestions of products to users (Koren 2008, 426; Rocca 2019). These algorithms aim to enhance user experience and build customer loyalty (Koren 2008, 426). They are commonly developed using content-based methods founded on user and product information and collaborative filtering methods based on user-product interactions (Rocca 2019).

In 2006, the streaming service company Netflix offered USD 1,000,000 to any team that managed to come up with an algorithm that improved on their internal recommendation system, Cinematch, by 10% or more. The hybrid team of KorBell, Big Chaos and Pragmatic Chaos - Bell-Kor's Pragmatic Chaos - triumphed the challenge (Van Buskirk 2009). The team utilized advanced methods of collaborative filtering, restricted Boltzmann machine and gradient-boosted decision trees, yet Koren (2008, 434; 2009, 9) of team KorBell emphasized the significance of content information from the baseline predictors in capturing the main biases in the dataset and refining the prediction of the algorithm.

Following this lead, this project aims to develop a modest recommendation system that is based on content information from baseline predictors in a similar dataset of the movie recommender MovieLens and that predicts future movie ratings of users with a root mean squared error (RMSE) rate of less than 0.86490. In the succeeding sections, we examine the dataset and user preferences then subsequently build algorithms to develop the recommendation system.

This undertaking is part of the capstone in the Professional Certificate Program in Data Science of Harvard Online.

MovieLens 10M Dataset

The MovieLens 10M Dataset was composed of 10,000,054 movie ratings from users with 20 ratings or more (GroupLens, n.d.; Harper and Konstan 2015). The dataset was downloaded and subjected to wrangling using an initial code in the language R provided by Harvard Online, which standardized its formatting and partitioning into 90% edx set and 10% final holdout test set across all projects.

The edx set was used to examine user preferences and to train and test content-based algorithms to develop the recommendation system. It contained 9,000,055 movie ratings, each with 6 variables: userId, movieId, rating, timestamp, title and genres (fig. 1). UserIds and movieIds were unique to each of its 69,878 users and 10,677 movies, respectively. Ratings ranged from 0.5 to 5 stars. Timestamps were rendered in Unix time. Titles and were captured manually according to how they appear in the movie database IMDb and included the years of movie release (GroupLens, n.d.).

On the other hand, the final holdout test set was reserved to evaluate the recommendation system. It contained 999,999 movie ratings, each with the aforementioned 6 variables. We circle back to this set in a later section.

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

Figure 1: First rows of the edx set.

User Preference

The interactions of the contents of the edx set were visualized using ggplot2 functions to examine user preferences.

MovieId and Genre

A plot of user ratings against movieIds of a random sample of the data showed an aggregration of the ratings on a number of movieIds (fig. 2). This indicated the tendency of users to rate certain movies more than others and consequently imparted which movies are popular.

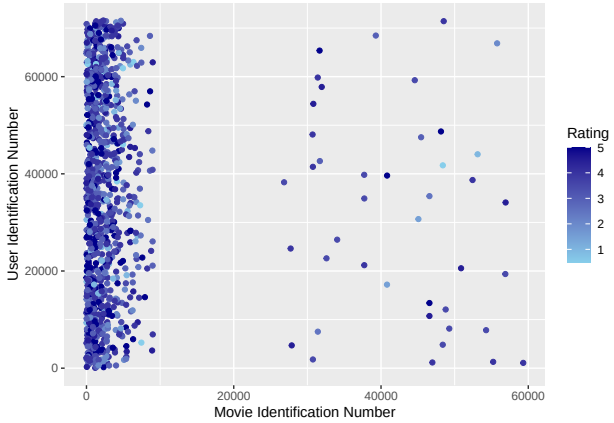

Figure 2: Scatterplot of user ratings vs movies of a random sample of the edx set.

A similar plot of user ratings against genres likewise exhibited the preference of users for certain genres such as Comedy and Drama (fig. 3).

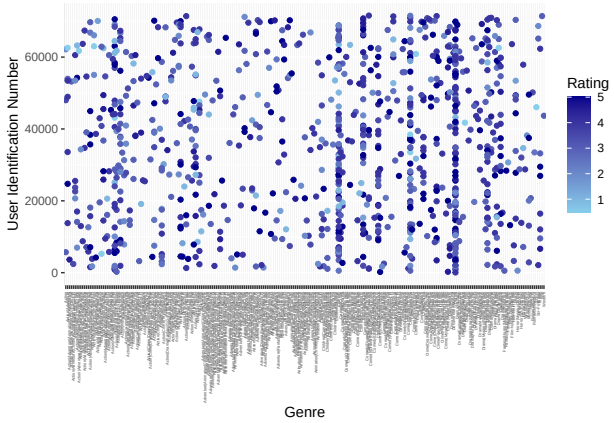

Figure 3: Scatterplot of user ratings vs genres of a random sample of the edx set.

Title and Year of Release

Titles were not unique to each movie unlike movieIds and were therefore not good movie identifiers. However, they contained important information: the years of movie release. Hence, the years were extracted from the ends of the titles utilizing stringr functions, and a plot of user ratings per year of a random sample of the data was produced (fig. 4). This revealed the inclination of users toward movies from the 1990s to 2000s.

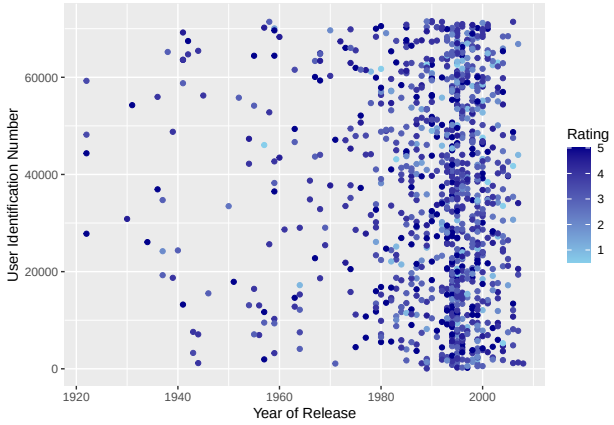

Figure 4: Scatterplot of user ratings per year of movie release of a random sample of the edx set.

UserId

A simple bar chart of average user ratings of a random sample of the data showed the tendencies for certain users to give a generous average rating of 5 (fig. 5). On the other hand, some users gave a stingy average rating of 1.

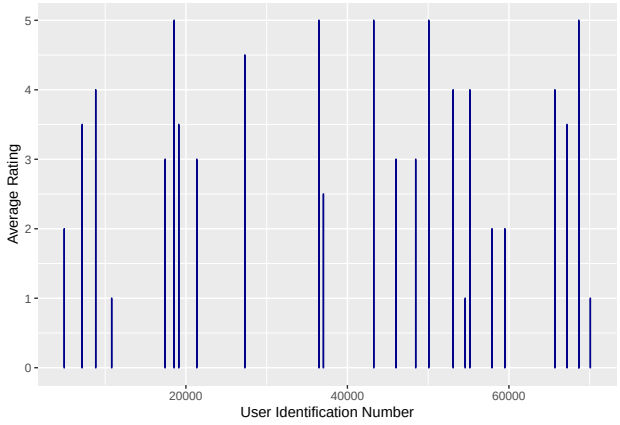

Figure 5: Bar chart of average user ratings of a random sample of the edx set.

Timestamp and Relative Age

Timestamps in Unix time implicitly contained the dates and times when the ratings were made. Using lubridate functions, the hour, day of the week, day of the month and month of the ratings were obtained. Plots of user ratings per hour, day of the week, day of the month and month of a random sample of the data were constructed but showed no readily observable bias (figs. 6-9).

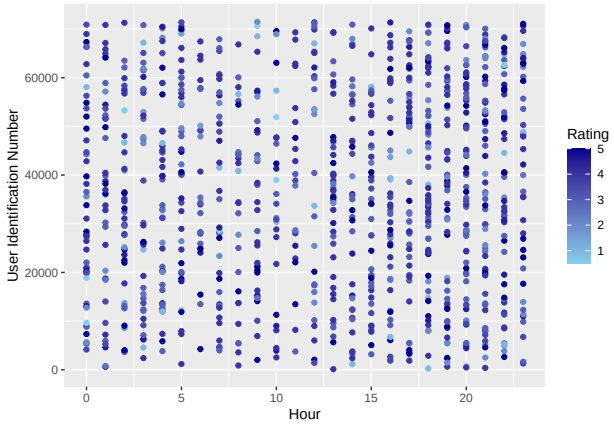

Figure 6: Scatterplot of user ratings per hour of the day of a random sample of the edx set.

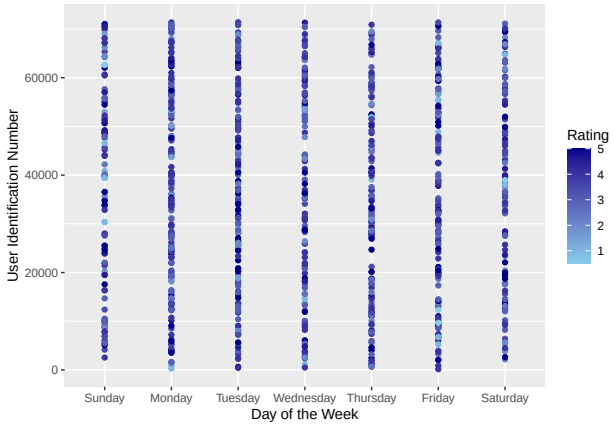

Figure 7: Scatterplot of user ratings per day of the week of a random sample of the edx set.

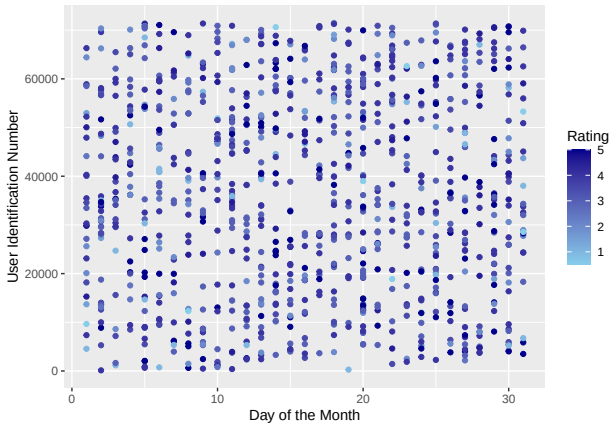

Figure 8: Scatterplot of user ratings per day of the month of a random sample of the edx set.

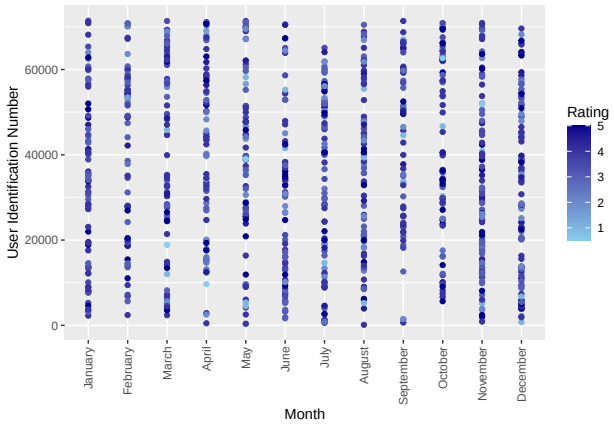

Figure 9: Scatterplot of user ratings per month of the year of a random sample of the edx set.

Unlike hours, days and months, the years of rating are not recurring units of time that we expect in future data. Instead of using them as is, the relative ages of movies were calculated from the years of rating and release through stringr and lubridate functions. A plot of user ratings against relative ages of a random sample of the data exhibited the preference of users to rate movies within 10 years of their release (fig. 10).

Figure 10: Scatterplot of user ratings vs relative ages of movies of a random sample of the edx set

Content-Based Algorithm

Given the biases observed in the contents of the edx set, the following were considered as possible predictors of the actual ratings in the recommendation system: movieId, genre, year of release, userId and relative age. Algorithms based on these predictors were progressively trained on 90% of the edx set (train set) by calculating the deviation caused by the predictor from the average using the penalized least squares equation (Irizarry 2022, Koren 2009). Then, they were tested on the remaining 10% of the edx set (test set) for the the RMSE of their predicted ratings from the actual ones. The RMSE was the preferred measure of error as it results in the same units as the given ones (Irizarry 2022).

Baseline Algorithm: Average Rating μ

The baseline algorithm was constructed on the simple assumption that the average rating μ of 3.5124 in the train set predicts ratings with the least error from the actual ratings in the testing set. This gave an RMSE

of 1.0593, which served as the basis of comparison for the RMSE of the succeeding algorithms.

```
mu <- mean(train_set$rating)
mu
## [1] 3.5124
baseline_rmse <- rmse(test_set$rating, mu)
baseline_rmse
## [1] 1.0593
```

Algorithm 1: Average Rating μ + Movie Bias b_i

The baseline algorithm predicted rating was improved by adding the deviation from the average caused by the observed movie bias. The bias of each movie b_i was calculated from the average of the movie ratings less the μ. This lowered the RMSE of the prediction to 0.94292.

```
bi_tibble <- train_set |>
  group_by(movieId) |>
  summarize(bi = mean(rating - mu))
head(bi_tibble, n = 5)
## # A tibble: 5 x 2
##   movieId     bi
##     <int>  <dbl>
## 1       1  0.419
## 2       2 -0.309
## 3       3 -0.366
## 4       4 -0.645
## 5       5 -0.442
algorithm1_rating <- test_set |>
  left_join(bi_tibble, by = "movieId") |> # adds the bi column
  mutate(algorithm1_rating = mu + bi) |>
  pull(algorithm1_rating)
algorithm1_rmse <- rmse(test_set$rating, algorithm1_rating)
algorithm1_rmse
## [1] 0.94292
```

Algorithm 2: Average Rating μ + Movie Bias b_i + Genre Bias b_g

The algorithm 1 predicted ratings were amended with the deviation from the average caused by the observed genre bias. The bias of each genre combination b_g was calculated from the average of the genre-combination ratings less the μ and b_i. However, this did not significantly affect the RMSE of the prediction, which remained at 0.94292.

```
bg_tibble <- train_set |>
  left_join(bi_tibble, by = "movieId") |>
  group_by(genres) |>
  summarize(bg = mean(rating - mu - bi))
head(bg_tibble, n = 5)
## # A tibble: 5 x 2
##   genres                                          bg
##   <chr>                                        <dbl>
## 1 (no genres listed)                         0
## 2 Action                                    -8.97e-17
```

```
## 3 Action|Adventure                                      -1.91e-15
## 4 Action|Adventure|Animation|Children|Comedy          -4.81e-16
## 5 Action|Adventure|Animation|Children|Comedy|Fantasy  4.47e-17
algorithm2_rating <- test_set |>
  left_join(bi_tibble, by = "movieId") |>
  left_join(bg_tibble, by = "genres") |>
  mutate(algorithm2_rating = mu + bi + bg) |>
  pull(algorithm2_rating)
algorithm2_rmse <- rmse(test_set$rating, algorithm2_rating)
algorithm2_rmse
## [1] 0.94292
```

Hence, the top genre was extracted from each genre combination with the str_detect function. Then, the b_g was recalculated from the average of the top-genre ratings less the μ and b_i. Still, this did not significantly affect the RMSE of the prediction, which remained at 0.94292.

```
genre_vector <- c("Action", "Adventure", "Animation", "Children", "Comedy", "Crime", "Documentary", "Dra
genre_rating <- sapply(genre_vector, function(g){
  train_set_a <- train_set |> filter(str_detect(genres, g) == TRUE)
  mean(train_set_a$rating)
})
genre_tibble <- tibble(genre = genre_vector, rating = genre_rating) |> arrange(-rating) # arranges the
genre_tibble
## # A tibble: 18 x 2
##    genre       rating
##    <chr>        <dbl>
##  1 Film-Noir     4.01
##  2 Documentary   3.78
##  3 War           3.78
##  4 Mystery       3.68
##  5 Drama         3.67
##  6 Crime         3.67
##  7 Animation     3.60
##  8 Musical       3.56
##  9 Western       3.55
## 10 Romance       3.55
## 11 Thriller      3.51
## 12 Fantasy       3.50
## 13 Adventure     3.49
## 14 Comedy        3.44
## 15 Action        3.42
## 16 Children      3.42
## 17 Sci-Fi        3.40
## 18 Horror        3.27
bg_tibble <- train_set |>
  left_join(bi_tibble, by = "movieId") |>
  mutate(t_genre = case_when(str_detect(genres, "Film-Noir") ~ "Film-Noir",
                             str_detect(genres, "Documentary") ~ "Documentary",
                             str_detect(genres, "War") ~ "War",
                             str_detect(genres, "Mystery") ~ "Mystery",
                             str_detect(genres, "Drama") ~ "Drama",
                             str_detect(genres, "Crime") ~ "Crime",
                             str_detect(genres, "Animation") ~ "Animation",
```

```
                            str_detect(genres, "Musical") ~ "Musical",
                            str_detect(genres, "Western") ~ "Western",
                            str_detect(genres, "Romance") ~ "Romance",
                            str_detect(genres, "Thriller") ~ "Thriller",
                            str_detect(genres, "Fantasy") ~ "Fantasy",
                            str_detect(genres, "Adventure") ~ "Adventure",
                            str_detect(genres, "Comedy") ~ "Comedy",
                            str_detect(genres, "Action") ~ "Action",
                            str_detect(genres, "Children") ~ "Children",
                            str_detect(genres, "Sci-Fi") ~ "Sci-Fi",
                            str_detect(genres, "Horror")) |>
  group_by(t_genre) |>
  summarize(bg = mean(rating - mu - bi))
head(bg_tibble, n = 5)
## # A tibble: 5 x 2
##   t_genre          bg
##   <chr>         <dbl>
## 1 Action     2.34e-16
## 2 Adventure  2.60e-15
## 3 Animation  3.45e-15
## 4 Children  -1.71e-16
## 5 Comedy     1.10e-15
algorithm2_rating <- test_set |>
  left_join(bi_tibble, by = "movieId") |>
  mutate(t_genre = case_when(str_detect(genres, "Film-Noir") ~ "Film-Noir",
                            str_detect(genres, "Documentary") ~ "Documentary",
                            str_detect(genres, "War") ~ "War",
                            str_detect(genres, "Mystery") ~ "Mystery",
                            str_detect(genres, "Drama") ~ "Drama",
                            str_detect(genres, "Crime") ~ "Crime",
                            str_detect(genres, "Animation") ~ "Animation",
                            str_detect(genres, "Musical") ~ "Musical",
                            str_detect(genres, "Western") ~ "Western",
                            str_detect(genres, "Romance") ~ "Romance",
                            str_detect(genres, "Thriller") ~ "Thriller",
                            str_detect(genres, "Fantasy") ~ "Fantasy",
                            str_detect(genres, "Adventure") ~ "Adventure",
                            str_detect(genres, "Comedy") ~ "Comedy",
                            str_detect(genres, "Action") ~ "Action",
                            str_detect(genres, "Children") ~ "Children",
                            str_detect(genres, "Sci-Fi") ~ "Sci-Fi",
                            str_detect(genres, "Horror") ~ "Horror")) |>
  left_join(bg_tibble, by = "t_genre") |>
  mutate(algorithm2_rating = mu + bi + bg) |>
  pull(algorithm2_rating)
algorithm2_rmse <- rmse(test_set$rating, algorithm2_rating)
algorithm2_rmse
## [1] 0.94292
```

Algorithm 3: Average Rating μ + Movie Bias b_i + Release Bias b_r

The algorithm 1 predicted ratings were updated with the deviation from the average caused by the observed year-of-release bias in place of the genre bias. The bias of each year of release b_r was calculated from the

average of the year-of-release ratings less the μ and b_i. However, this did not significantly affect the RMSE of the prediction as well, which remained at 0.94292.

```
br_tibble <- train_set |>
  left_join(bi_tibble, by = "movieId") |>
  mutate(year_rel = str_extract(title, "\\(\\d{4}\\)$")) |>
  mutate(year_rel = str_replace_all(year_rel, "[:punct:]", "")) |>
  mutate(year_rel = as.integer(year_rel)) |>
  group_by(year_rel) |>
  summarize(br = mean(rating - mu - bi))
head(br_tibble, n = 5)
## # A tibble: 5 x 2
##   year_rel        br
##      <int>     <dbl>
## 1     1915 -5.71e-17
## 2     1916 -5.48e-17
## 3     1917 -1.53e-17
## 4     1918 -3.22e-18
## 5     1919  2.25e-17
algorithm3_rating <- test_set |>
  left_join(bi_tibble, by = "movieId") |>
  mutate(year_rel = str_extract(title, "\\(\\d{4}\\)$")) |>
  mutate(year_rel = str_replace_all(year_rel, "[:punct:]", "")) |>
  mutate(year_rel = as.integer(year_rel)) |>
  left_join(br_tibble, by = "year_rel") |>
  mutate(algorithm3_rating = mu + bi + br) |>
  pull(algorithm3_rating)
algorithm3_rmse <- rmse(test_set$rating, algorithm3_rating)
algorithm3_rmse
## [1] 0.94292
```

Algorithm 4: Average Rating μ ***+ Movie Bias*** b_i ***+ User Bias*** b_u

Algorithm 5: Average Rating μ ***+ Movie Bias*** b_i ***+ User Bias*** b_u ***+ Age Bias*** b_a

Algorithm 6: Average Rating μ ***+ Regularized Movie Bias*** rb_i ***+ User Bias*** rb_u ***+ Age Bias*** rb_a

<TABLE 1> (Madhugiri 2022)

"Explainability is another key point of the success of recommendation algorithms. Indeed, it has been proven that if users do not understand why they had been recommended a specific item, they tend to lose confidence in the recommender system." (Rocca 2019)

"In order to combat overfitting the sparse rating data, models are regularized so estimates are shrunk towards baseline defaults. Regularization is controlled by constants, which are denoted as: lambda1, lambda2, . . . " (Koren 2008, 427)

"The regularizing term lambda avoid overfitting by penalizing magnitudes of the parameters." (Koren 2009, 2)

Recommendation System

Conclusion

The lm function offers a convenient way of fitting this linear model but will be time and space consuming in this case.

NOTE: we use the penalized least squares equation and not train() to train algorithms with as we are not constrained by time and/or computer capability

"In this paper, we suggested methods that lower the RMSE to 0.8870."

"However, they are entered manually, so errors and inconsistencies may exist." (GroupLens, n.d.)

<LIMIT: RECURRING TIME>

"in others our success is restricted by the randomness of the process, with movie recommendations for example." (Irizarry 2022)

Reference

GroupLens, n.d. "README.txt." Accessed May 29, 2023. https://grouplens.org/datasets/movielens/10m/.

Harper, F. Maxwell, and Joseph A. Konstan. 2015. "The MovieLens Datasets: History and Context." *ACM Transactions on Interactive Intelligent Systems* 5, no. 4: 1-19. https://doi.org/10.1145/2827872.

Irizarry, Rafael A. 2002. *Introduction to Data Science: Data Analysis and Prediction Algorithms with R.* http://rafalab.dfci.harvard.edu/dsbook/.

Koren, Yehuda. 2008. "Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model." https://people.engr.tamu.edu/huangrh/Spring16/papers_course/matrix_factorization.pdf.

Koren, Yehuda. 2009. "The BellKor Solution to the Netflix Grand Prize." https://www2.seas.gwu.edu/~simhaweb/champalg/cf/papers/KorenBellKor2009.pdf.

Madhugiri, Devashree. 2022. "Top 7 Packages for Making Beautiful Tables in R." https://towardsdatascience.com/top-7-packages-for-making-beautiful-tables-in-r-7683d054e541.

Rocca, Baptiste. 2019. "Introduction to Recommender Systems." https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada.

Van Buskirk, Eliot. 2009. "BellKor's Pragmatic Chaos Wins $1 Million Netflix Prize by Mere Minutes." https://www.wired.com/2009/09/bellkors-pragmatic-chaos-wins-1-million-netflix-prize/.