HarvardX Capstone Project: Movie Recommendation Systems

Francis Dzakpasu

2025-06-28

Summary

Movie recommendation systems are filtering algorithms online streaming services often use to predict users' movie preferences. This document used data from MovieLens to create a movie recommendation system. The MovieLen 10M dataset, was partitioned into two sets – 'edx' set for training the recommendation system algorithm and a 'final hold-out test' set for final evaluation of the trained algorithm using 'root mean square error – RMSE' loss function. First, the dataset (edx set) structure was explored using descriptive statistics and visualisations. The algorithm training utilised two different recommendation system machine learning techniques, linear regression (with and without regularisation) and matrix factorisation systems, to predict movie rating outcomes based on movies and users features (predictors). Test performance of the trained recommendation systems showed the matrix factorisation system performed better in the movie predictions ($RMSE = 0.786$) than the linear regression with regularisation system ($RMSE = 0.865$) and the linear regression system ($RMSE = 0.866$). Although other relevant attributes of the MovieLens dataset such as the movies' genres, movies' released year, etc., were unaccounted for in the models, the matrix factorisation seems to be more robust in accounting for the complex dimensionality in the dataset.

Introduction

Machine learning algorithms are utilised in recommendation systems and have become integral to online businesses such as e-commerce and other online streaming merchandise like Netflix, Amazon, etc.[1] Recommendation systems are information filtering systems that predict users' preferences on products or topics[2]. These reduce time-consuming searches, provide easy content access, and enable businesses to improve customer experience simultaneously[3]. Importantly, recommendation systems improve businesses' competitive edge, retain customers and reduce the rate of customers leaving for other competitors when they see that their needs and demands are understood and delivered at all times[4].

Recommendation systems have become a common online experience, for example, streaming services such as YouTube automatically play clips related to those a user watched or suggested what they might like[5]. Customers on Amazon often see automatically suggested products that are bought together or based on

[1]Khanal SS, Prasad PWC, Alsadoon A, Maag A: A systematic review: machine learning based recommendation systems for e-learning. Education and Information Technologies 2020, 25(4):2635-2664.

[2]Tran DT, Huh J-H: New machine learning model based on the time factor for e-commerce recommendation systems. The Journal of Supercomputing 2023, 79(6):6756-6801.

[3]Rathor K, Chandre S, Thillaivanan A, Raju MN, Sikka V, Singh K: Archimedes Optimization with Enhanced Deep Learning based Recommendation System for Drug Supply Chain Management. In: 2023 2nd International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN): 21-22 April 2023: 1-6.

[4]Loukili M, Messaoudi F, El Ghazi M: Machine learning based recommender system for e-commerce. IAES International Journal of Artificial Intelligence 2023, 12(4):1803-1811.

[5]Covington P, Adams J, Sargin E: Deep Neural Networks for YouTube Recommendations. In: Proceedings of the 10th ACM Conference on Recommender Systems. Boston, Massachusetts, USA: Association for Computing Machinery; 2016: 191–198.

their purchase history[6]. There are many more examples of recommendation systems, including Facebook users seeing suggested new friends and subscribers on Netflix seeing automatically suggested movies[7].

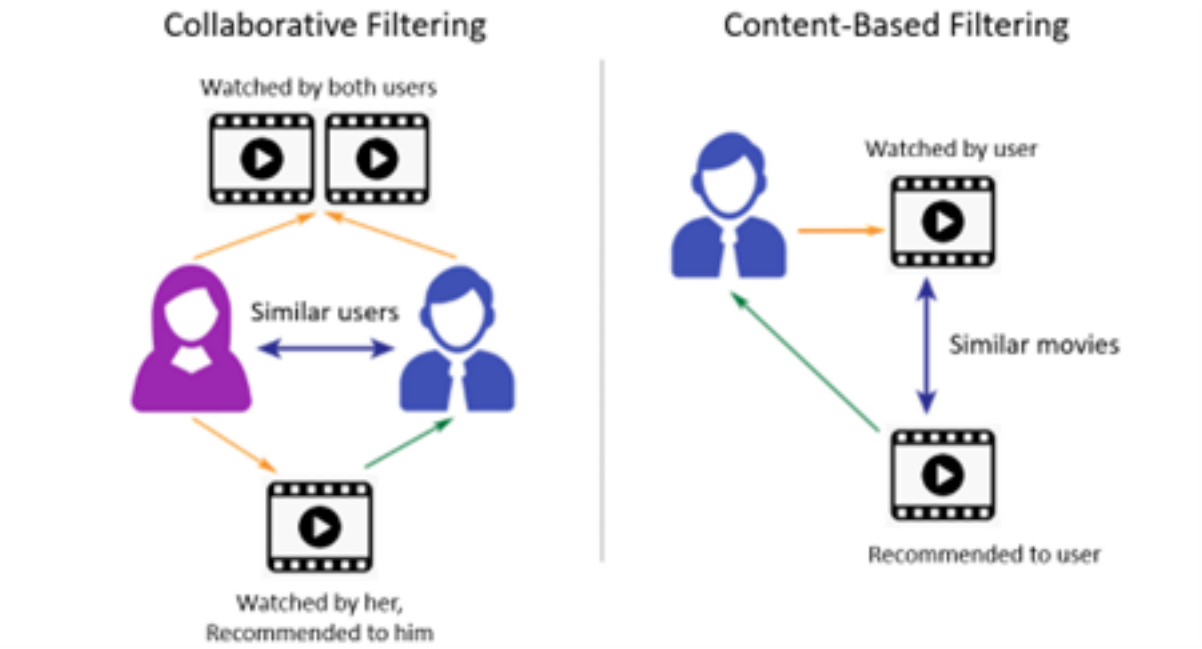


Figure 1: **An illustration of a Content-based and collaborative filtering recommendation system.** *(Note: The image is taken from Divyesh Prajapati[9])*

Movie recommendation systems filter or predict users' possible movie choices based on their past choices or history of movie buying behaviour. Typically, two types of data are used in movie recommendation systems, i.e., the rating or buying behaviours that reflect the attributable information about the movies, users, and keywords or textural profiles that record the user-movie interactions[10]. According to the inference data, movie recommendation systems commonly use content-based and collaborative filtering recommendation systems[11]. Collaborative filtering models analyse commonalities and similarities among users based on the ratings and then estimate new recommendations according to user relationships. By contrast, content-based filtering models use the contributions of an item to recommend other items similar to the user's preferences. They are mainly concerned with rating one user instead of multiple users[12].

This capstone document aims to create a movie recommendation system using the MovieLens dataset and evaluate the quality of predictions by calculating the Root Mean Square Error (RMSE).

[6]Gómez-Losada Á, Duch-Brown N: Time Series Forecasting by Recommendation: An Empirical Analysis on Amazon Marketplace. In: Business Information Systems: 2019// 2019; Cham: Springer International Publishing; 2019: 45-54.

[7]Gomez-Uribe CA, Hunt N: The Netflix Recommender System: Algorithms, Business Value, and Innovation. ACM Trans Manage Inf Syst 2016, 6(4):Article 13.

[9]Prajapati D: Introduction to Movie Recommendation System for Beginners. Medium, 2021. URL: https://divyeshprajapati100.medium.com/introduction-to-movie-recommendation-system-for-beginners-a12987d94adc

[10]Mu Y, Wu Y: Multimodal Movie Recommendation System Using Deep Learning. In: Mathematics. vol. 11; 2023.

[11]Aggarwal CC: Recommender systems, vol. 1: Springer; 2016.

[12]Lu J, Wu D, Mao M, Wang W, Zhang G: Recommender system application developments: A survey. Decision Support Systems 2015, 74:12-32.

Methods

Overview of the MovieLens datasets

The MovieLens datasets describe people's preferences for movies which the GroupLens Research manages at the Univeristy of Minnesota. The preferences use the form of tuples, in which a person expresses a rating (0–5 stars) for a watched movie at a particular time. The rating data in MovieLens include timestamps and explicit feedback[13]. The complete dataset consists of about 27 million ratings of 58,000 movies by 280,000 users. This document considers a subset dataset, MovieLens 10M – a stable benchmark dataset, with 10 million ratings on 10,000 movies by 72,000 users. The variables in the dataset include:

- userId: the user's ID
- movieId: the movie ID
- rating: the user-rated score of the movie
- timestamp: the time a movie rated by a user
- title: the movie title
- genres: the movie genre

MovieLens 10M datasets download and cleaning

The Movielens datasets, included ratings.dat and movie.dat were downloaded and loaded into the workspace. The required packages were installed and the libraries loaded.

```
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")
if(!require(Metrics)) install.packages("Metrics",
                                       repos = "http://cran.us.r-project.org")
if(!require(kableExtra)) install.packages("kableExtra",
                                          repos = "http://cran.us.r-project.org")
if(!require(recosystem)) install.packages("recosystem",
                                          repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(broom)
library(lubridate)
library(gam)
library(randomForest)
library(Rborist)
library(matrixStats)
library(rafalib)
library(naivebayes)
library(ggplot2)
library(ggthemes)
library(Metrics)
```

[13]Kane F: Building Recommender Systems with Machine Learning and AI: Help people discover new products and content with deep learning, neural networks, and machine learning recommendations: Independently published; 2018.

```
library(scales)
library(dslabs)
ds_theme_set()
library(knitr)
library(kableExtra)
library(recosystem)
```

The downloading, cleaning and merging of the rating.dat and movie.dat into one DataFrame is shown below.

```
## MovieLens 10M dataset:
#  https://grouplens.org/datasets/movielens/10m/
#  http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"),
                         simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"),
                        simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

# Merging the two data frames into one MovieLens data frame
movielens <- left_join(ratings, movies, by = "movieId")
```

The 'edx' and 'final hold-out test' data sets

The merged MovieLens DataFrame was split into two sets, a "final hold-out test" set, which made up 10% of the MovieLens dataset for the evaluation of the the final built recommendation system algorithm, and a "edx" set for building the algorithm.

```r
# Splitting the MovieLens dataset to create the final evaluation set (final hold-out test)
# Final hold-out test set would be 10% of MovieLens data

set.seed(1, sample.kind="Rounding") # if using R 3.6 or later

# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)

## Save the cleaned data - final hold-out test and edx sets
# final_holdout_test - ONLY be used for evaluating the RMSE of your final algorithm
save(final_holdout_test, file = "final_holdout_test.RData")

# edx data - training and testing sets and/or use cross-validation to design and
# test algorithm
save(edx, file = "edx.RData")
```

Data exploration

The dataset (edx set) was first explored using summary statistics and visualisations to familarise with the basic structure.

```r
load("final_holdout_test.RData")
load("edx.RData")
```

```r
# The class of the dataset
class(edx)
```

```
## [1] "data.frame"
```

```r
# The structure
str(edx, vec.len = 2)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 ...
##  $ rating   : num  5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" ...
```

```
# Examining the first few rows
head(edx)
```

```
##   userId movieId rating timestamp                          title
## 1      1     122      5 838985046                Boomerang (1992)
## 2      1     185      5 838983525                 Net, The (1995)
## 4      1     292      5 838983421                 Outbreak (1995)
## 5      1     316      5 838983392                 Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1                Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

```
# The summary of the edx set
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The number of unique users ratings and the unique movies rated.

```
# The number of unique users ratings and the unique movies rated
n_users_movies <- edx %>%
  summarize(tibble(unique_users = n_distinct(userId),
                   unique_movies = n_distinct(movieId)))
```

The most common genre and average ratings

```
## The most common genre and average ratings
edx %>%
  separate_rows(genres,
                sep = "\\|") %>%
  group_by(genres) %>%
  summarize(n = n(),
            avg_ratings = mean(rating)) %>%
  arrange(desc(n))
```

```
## # A tibble: 20 x 3
##    genres                  n avg_ratings
##    <chr>               <int>       <dbl>
##  1 Drama             3910127        3.67
##  2 Comedy            3540930        3.44
##  3 Action            2560545        3.42
##  4 Thriller          2325899        3.51
##  5 Adventure         1908892        3.49
##  6 Romance           1712100        3.55
##  7 Sci-Fi            1341183        3.40
##  8 Crime             1327715        3.67
##  9 Fantasy            925637        3.50
## 10 Children           737994        3.42
## 11 Horror             691485        3.27
## 12 Mystery            568332        3.68
## 13 War                511147        3.78
## 14 Animation          467168        3.60
## 15 Musical            433080        3.56
## 16 Western            189394        3.56
## 17 Film-Noir          118541        4.01
## 18 Documentary         93066        3.78
## 19 IMAX                 8181        3.77
## 20 (no genres listed)      7        3.64
```

Also, knowing the genres associated with the movies in the MovieLens dataset, the number of ratings for the individual genres can be determined as follows:

```
## Number of ratings per genres can be determined as follows:
genres <- c("Action", "Adventure", "Animation", "Children", "Comedy", "Crime",
            "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror", "IMAX",
            "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western")
n_ratings <- function(x) {
  sum(str_detect(edx$genres, x))
}
tibble(Genres = genres,
       Count = unlist(lapply(genres, n_ratings)))
```

```
## # A tibble: 19 x 2
##    Genres        Count
##    <chr>         <int>
##  1 Action      2560545
##  2 Adventure   1908892
##  3 Animation    467168
##  4 Children     737994
##  5 Comedy      3540930
##  6 Crime       1327715
##  7 Documentary   93066
##  8 Drama       3910127
##  9 Fantasy      925637
## 10 Film-Noir    118541
## 11 Horror       691485
## 12 IMAX           8181
## 13 Musical      433080
```

```
## 14 Mystery      568332
## 15 Romance     1712100
## 16 Sci-Fi      1341183
## 17 Thriller    2325899
## 18 War          511147
## 19 Western      189394
```

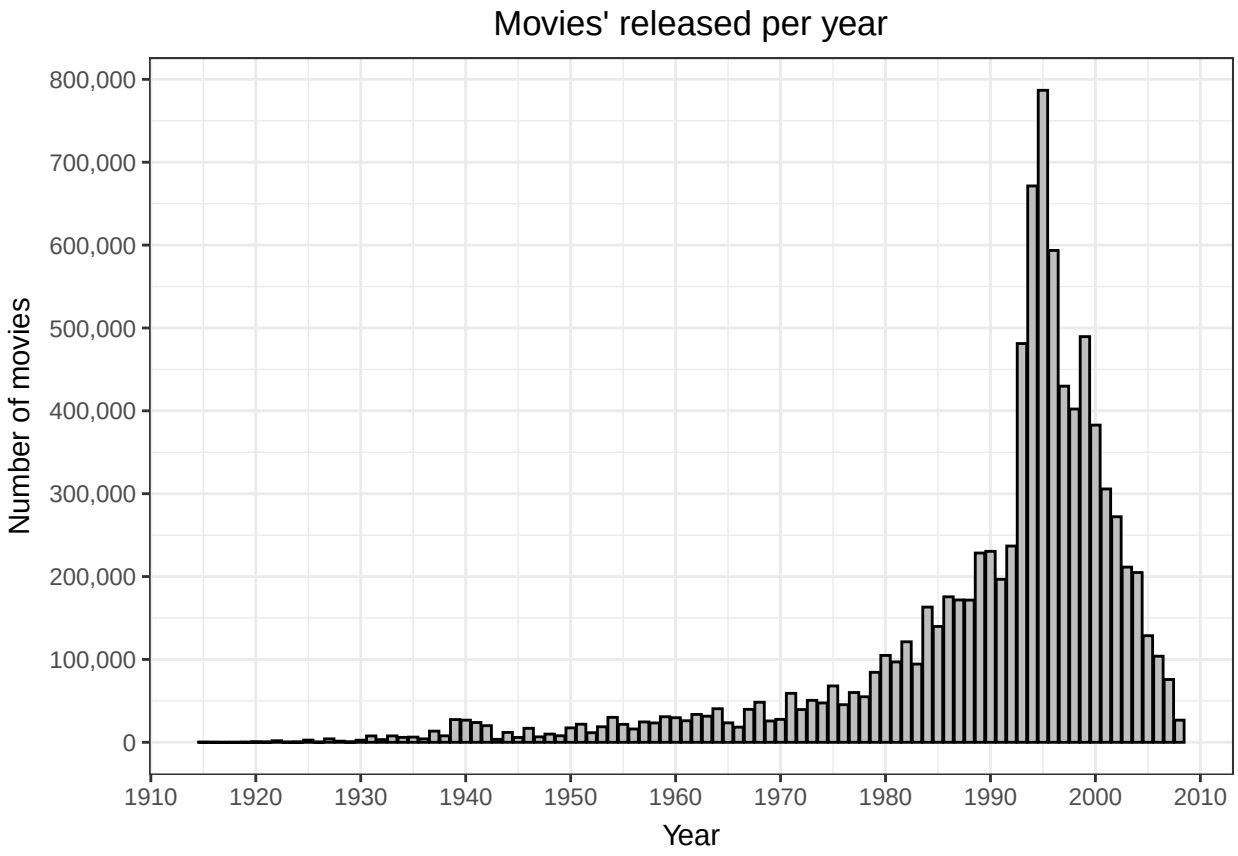
The most common ratings score for a movie is:

```
## The common rating score
edx %>%
  group_by(rating) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    4   2588430
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
##  6    4.5  526736
##  7    1    345679
##  8    2.5  333010
##  9    1.5  106426
## 10    0.5   85374
```

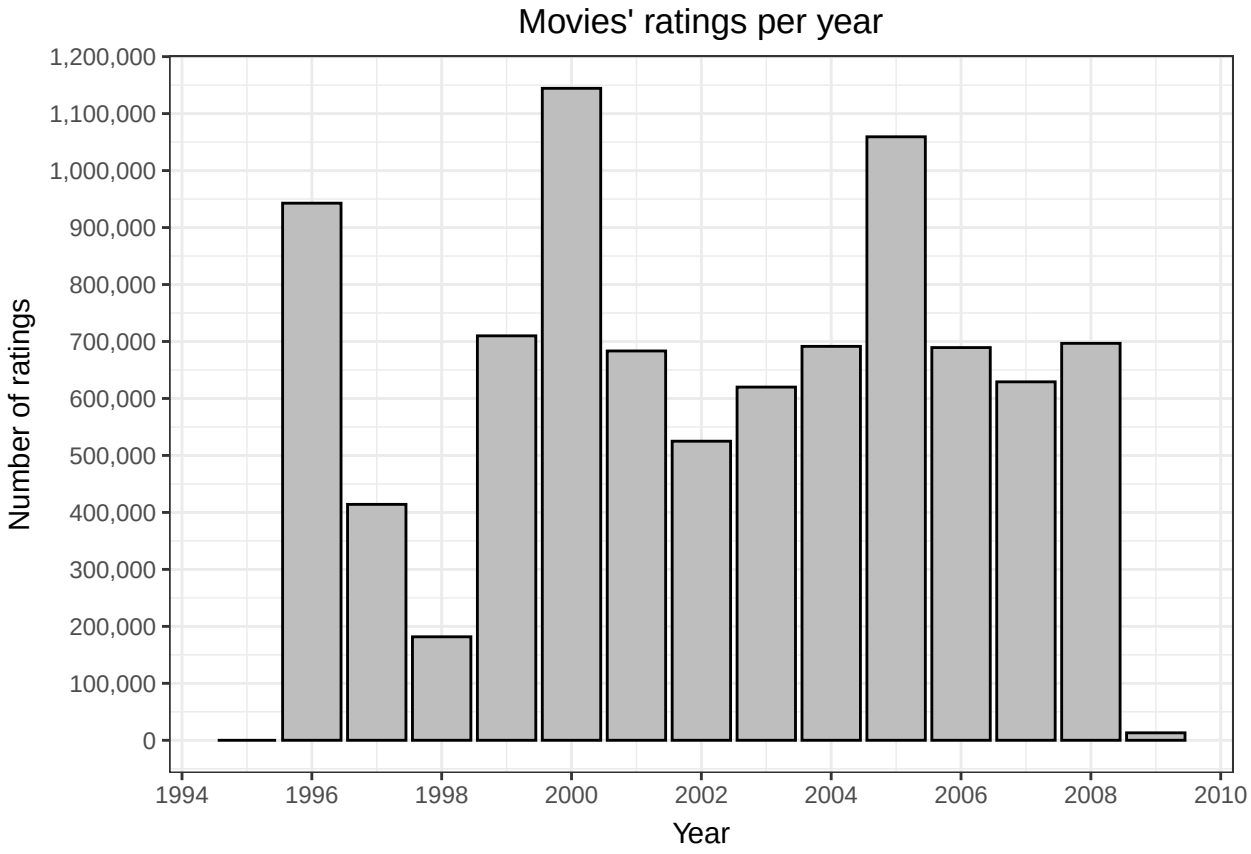
The total number of movies released per year is illustrated with a bar plot below.

```
## A plot of total number of movies released per year
edx  %>%
  mutate(year = as.numeric(str_extract(str_extract(title, "[/(]\\d{4}[/)]$"),
                                       regex("\\d{4}")))) %>%
  group_by(year) %>%
  mutate(n=n()) %>%
  ggplot(aes(year)) +
  geom_bar(fill = "grey", color = "black") +
  ggtitle("Movies' released per year") +
  xlab("Year") +
  ylab("Number of movies") +
  scale_y_continuous(n.breaks = 10, labels = scales::label_comma()) +
  scale_x_continuous(n.breaks = 10) +
  theme_bw() +
  theme(axis.title.x = element_text(vjust = 0),
        axis.title.y = element_text(vjust = 2),
        plot.title = element_text(hjust=0.5))
```



The total number of movies rated each year is also ploted.

```
edx %>%
  mutate(year = year(as_datetime(timestamp, origin = "1970-01-01"))) %>%
  ggplot(aes(x = year)) +
  geom_bar(fill = "grey", color = "black") +
  ggtitle("Movies' ratings per year") +
  xlab("Year") +
  ylab("Number of ratings") +
  scale_y_continuous(n.breaks = 10, labels = scales::label_comma()) +
  scale_x_continuous(n.breaks = 10) +
  theme_bw() +
  theme(axis.title.x = element_text(vjust = 0),
        axis.title.y = element_text(vjust = 2),
        plot.title = element_text(hjust=0.5))
```



Histogram visualisations exploring the users' and movies' ratings

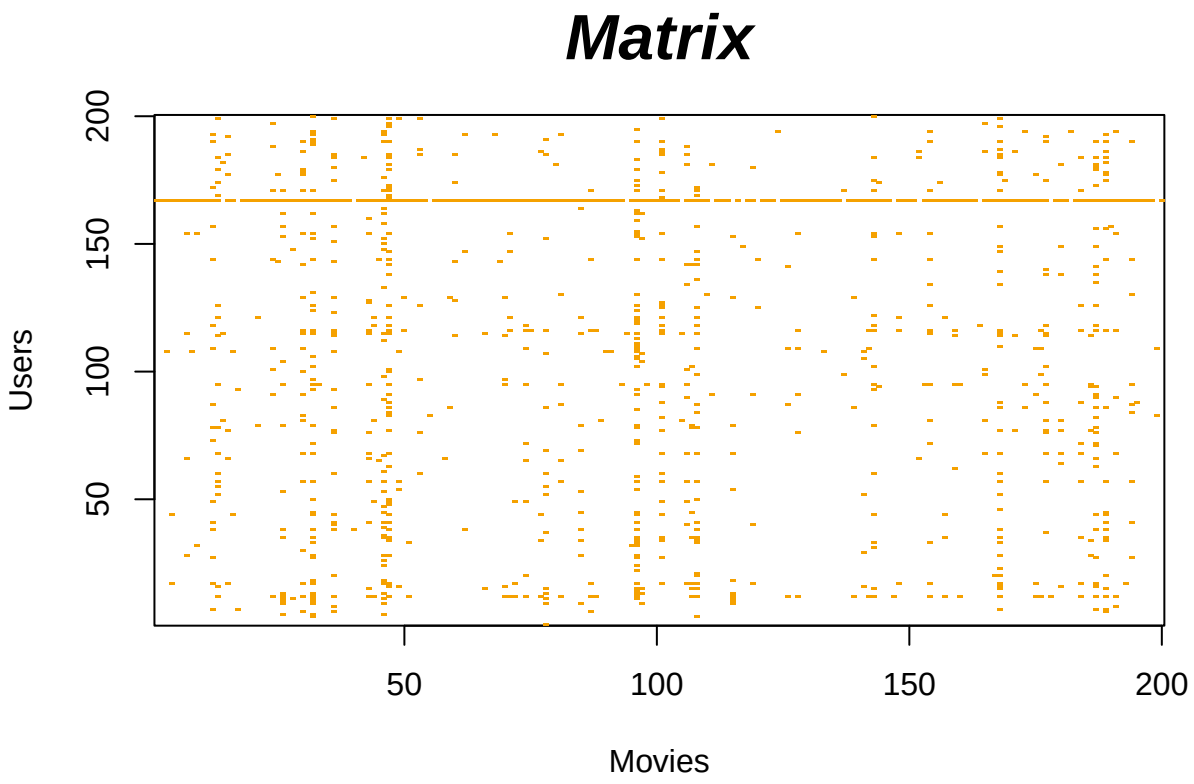
```
# Users' ratings
hist_users <- edx %>%
  group_by(userId) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = count)) +
  geom_histogram(fill = "grey", color = "black") +
  ggtitle("Users' ratings") +
  xlab("Rating count") +
  ylab("Number of users") +
  scale_y_continuous(n.breaks = 10) +
  scale_x_log10(n.breaks = 5) +
  theme_bw() +
  theme(axis.title.x = element_text(vjust = 0),
        axis.title.y = element_text(vjust = 2),
        plot.title = element_text(hjust=0.5))

## Movies' ratings
hist_movies <-edx %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = count)) +
  geom_histogram(width="40", fill = "grey", color = "black") +
  ggtitle("Movies' ratings") +
  xlab("Rating count") +
```

```
  ylab("Number of movies") +
  scale_y_continuous(n.breaks = 10) +
  scale_x_log10(n.breaks = 5) +
  theme_bw() +
  theme(axis.title.x = element_text(vjust = 0),
        axis.title.y = element_text(vjust = 2),
        plot.title = element_text(hjust=0.5))

gridExtra::grid.arrange(hist_users, hist_movies, ncol = 2)
```

User/movie combination Matrix: A matrix is constructed for randomly selected users to illustrate the user/movie combinations that have rating data.

```
N <- 200
matrix_const <- edx %>%
  filter(userId %in% sample(unique(edx$userId), N)) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), N)) %>%
  as.matrix() %>%
  t(.) # to transposes the matrix

# Matrix plot
```

```r
matrix_const %>%
  image(1:N, 1:N,. , xlab="Movies", ylab="Users")
#abline(h=0:N+0.5, v=0:N+0.5, col = "grey")
title(main = list("Matrix", cex = 2, font = 4))
```



NB: the yellow spots indicate user/movie combinations with rating.

The Recommendation Systems

In order to validate the algorithms building process, the edx dataset was partitioned into two sets, a train_edx and test_edx sets, with the test_edx being 10% of the data reserved for validating the performance throughout the algorithms training.

```r
# Set the seed to 1
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_edx <- edx[-test_index,]
test_data <- edx[test_index,]

# Ensuring userId and movieId in the training set are also in testing set
test_edx <- test_data %>%
  semi_join(train_edx, by = "movieId") %>%
  semi_join(train_edx, by = "userId")
```

```r
# Adding rows removed from the testing set back into the training set
removed <- anti_join(test_data, test_edx)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

```r
train_edx <- rbind(train_edx, removed)

## Number of observations in the partitioned data sets
nrow(train_edx)
```

```
## [1] 8100065
```

```r
nrow(test_edx)
```

```
## [1] 899990
```

Root mean square error

Root mean square error (RMSE) is the commonly used loss function to evaluate the quality of predictions in Machine Learning. It measures how far predicted values deviate from observed values in regression models[14]. This is computed by calculating the residual for each data point, computing the residual norm for each data point, computing the mean of residuals, and taking the square root of the computed mean. This is presented mathematically as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{u,i}\left(\hat{Y}_{u,i} - Y_{u,i}\right)^2}$$

Where $Y_{u,i}$ is the rating for movie i by user u, whereas $\hat{Y}_{u,i}$ is the predicted rating and N is the number of user/movie combinations.

Note that the smaller the calculated $RMSE$, the better the quality of the movie prediction.

RMSE function: The code for RMSE function is shown below:

```r
## Model Evaluation function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Building the prediction algorithms This document utilised two machine learning techniques to train the movie recommendation system, including a linear regression method (with and without regularisation) and a more robust matrix factorisation method.

1. **Linear regression model**

[14]Sharma DK, Chatterjee M, Kaur G, Vavilala S: 3 - Deep learning applications for disease diagnosis. In: Deep Learning for Medical Applications with Unique Data. edn. Edited by Gupta D, Kose U, Khanna A, Balas VE: Academic Press; 2022: 31-51.

Linear regression modelling in machine learning algorithms is often a baseline method and works well with some challenges[15]. Generally, linear regression for outcome Y and predictor X is expressed as:

$$E(Y|X = x)_i = \beta_0 + \beta_{x_i} + \varepsilon_i; i = 1, 2, 3, ..., N$$

The predicted outcome $\hat{Y}|X = x$ is given by the expression:

$$\hat{Y}|X = x_i = \mu + b_1 x_i + \varepsilon_i; i = 1, 2, 3, ..., N$$

The presented linear regression recommendation system used movie ratings as the outcome (Y) and the predictors (features) were 'movieId' (i) and 'userId' (u).

a. **The base model**

The predicted rating is based on the mean rating for the movies without taking into consideration the effects of the individual features; thus, the movies and users effects. The regression equation looks like this:

$$\hat{Y}_{u,i} = \mu + \varepsilon_{u,i}$$

where μ is the mean rating for all movies by all users and $\varepsilon_{i,u}$ a normally distributed independent errors of the sample. The mean rating μ was calculated and the predicted $\hat{Y}$ estimated.

```
mu <- mean(train_edx$rating)
y_hat_base <- rep(mu, nrow(test_edx))
```

The RMSE was calculated to examine the quality of the base model prediction.

```
base_model_rmse <- RMSE(y_hat_base, test_edx$rating)
base_model_rmse
```

```
## [1] 1.060054
```

```
kable(rmse_evaluations <- tibble(Method = "Linear regression: base model",
                                 RMSE = base_model_rmse))
```

Note that the calculated RMSE is greater than one (>1), indicating that the quality of the base linear regression model predicting a movie rating is low. The prediction error is likely greater than a 1-star rating, making it a bad model[16]. Therefore, the predictors (movies and users) were introduced in the model to account for their effects on the prediction.

b. **Movies and users effects models**

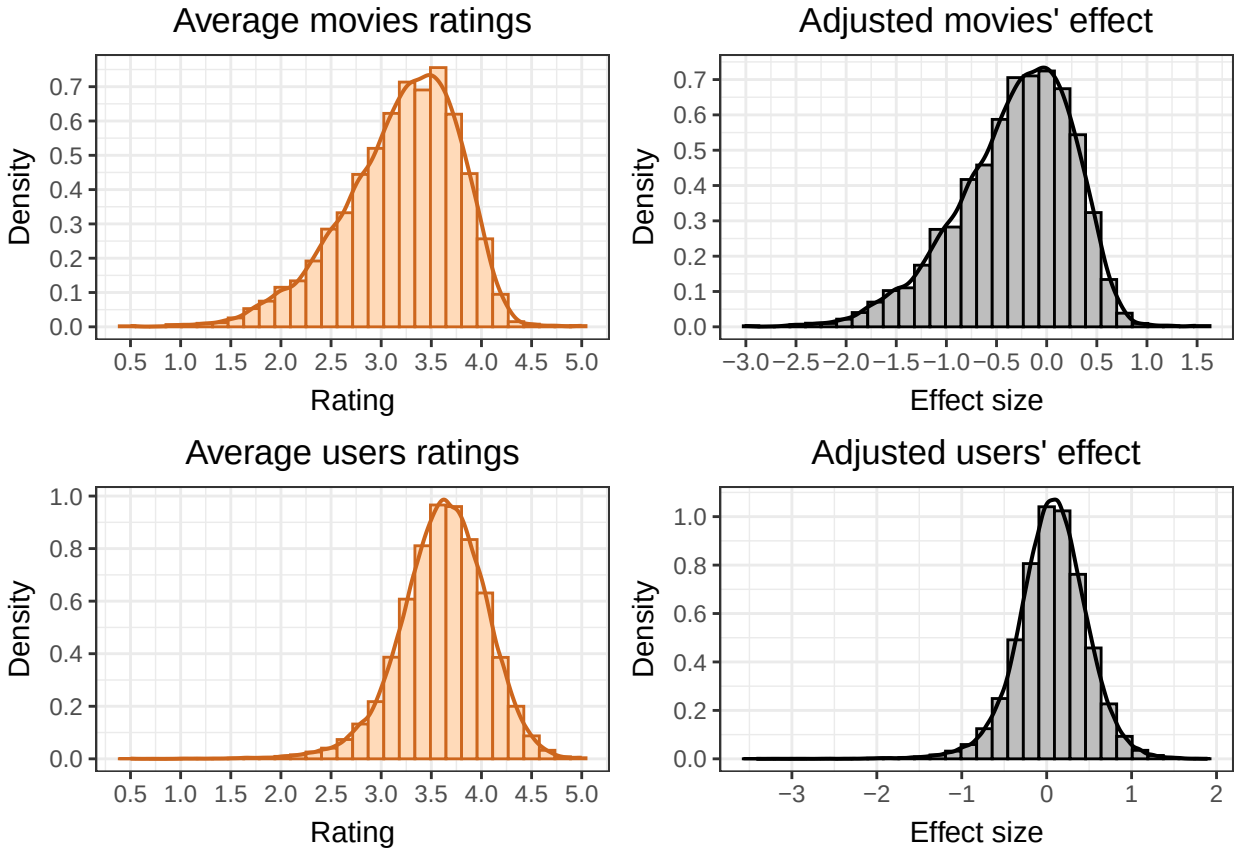
To account for the potential bias of the movies and users in the prediction, the movies (b_i) and users (b_u) effects size were first calculated by the least squares estimate approach.

[15]Irizarry R. A: Introduction to Data Science. Data Analysis and Prediction Algorithms with R. 2024 url: https://rafalab.dfci.harvard.edu/dsbook/

[16]Irizarry R. A: Introduction to Data Science. Data Analysis and Prediction Algorithms with R. 2024 url: https://rafalab.dfci.harvard.edu/dsbook/

```r
## Calculate the movies' mean rating and the adjusted effect
movie_bias <- train_edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu),  # b_i = adjusted movie effect
            b_i0 = mean(rating))      # b_i0 = average movies' ratings

## Calculate the users' mean rating and the adjusted effect
user_bias <- train_edx %>%
  left_join(movie_bias, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - (mu+b_i)), # b_u = adjusted user effect
            b_u0 = mean(rating))           # b_u0 = average users' ratings
```

A graphical illustration of the adjusted movies and users effects and the average ratings for movies and users are presented to illustrate the visual variations in the movies rating.



Adjusted linear regression models

Accounting for movies effects: First and foremost, given that the individual movies were rated differently which could potentially bias the base model, the movies' effect was accounted for in the linear regression model. The prediction equation for the adjusted movies effects linear regression model is given as:

$$\hat{Y}_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

where b_i is the adjusted movies effected.

The predicted ratings $\hat{Y}$ and the RMSE are shown below:

```
## Model with only movie effect
y_hat_movie <- mu + test_edx %>%
  left_join(movie_bias, by = "movieId") %>%
  pull(b_i)

movie_only_rmse <- RMSE(y_hat_movie, test_edx$rating)
movie_only_rmse
```

```
## [1] 0.9429615
```

```
kable(rmse_evaluations <- bind_rows(rmse_evaluations,
                    tibble(Method="Linear regression: with movies' effects",
                                                RMSE = movie_only_rmse )))
```

Accounting for users effect model: Furthermore, the individual users rated the movies differently; thus, there are potential variations in the users rating which also need to be accounted for. The accounted users' effect predictions model shown in the equation below:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

where b_u is the adjusted user-specific effect.

The predicted ratings $\hat{Y}$ with accounted movies' and users' effects and the RMSE are as follows:

```
## Model with adjusted movies' and users' effect
y_hat_adjusted <- test_edx %>%
  left_join(movie_bias, by='movieId') %>%
  left_join(user_bias, by='userId') %>%
  mutate(y_hat_adjusted = mu+b_i+b_u) %>%
  pull(y_hat_adjusted)

adjusted_effects_rmse <- RMSE(y_hat_adjusted, test_edx$rating)
adjusted_effects_rmse
```

```
## [1] 0.8646843
```

```
kable(rmse_evaluations <- bind_rows(rmse_evaluations,
                  tibble(Method="Linear regression: adjusted movies' & users' effects",
                                                RMSE = adjusted_effects_rmse)))
```

To check the performance of the trained linear regression recommendation algorithm, the model was applied to the final hold-out data set to predict movie ratings and the RMSE was calculated.

```
## Movies prediction
movie_rating_finalhold <- final_holdout_test %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  mutate(y_hat_movie_recomd = mu+b_i+b_u) %>%
```

```
  pull(y_hat_movie_recomd)

movie_rating_finalhold_rmse <- RMSE(movie_rating_finalhold, final_holdout_test$rating)
movie_rating_finalhold_rmse
```

```
## [1] 0.8658528
```

```
kable(rmse_evaluations <- bind_rows(rmse_evaluations,
    tibble(Method="Linear regression: movies' & users' effects on final hold-out test",
                                RMSE = movie_rating_finalhold_rmse)))
```

The trained adjusted linear regression was then applied to the final hold-out test set to make movie predictions that are shown in the Results section. The codes for the best and worst movie predictions are presented below.

```
## Best movies recommended by the linear model on the final_holdout_test set
best20_movies <- final_holdout_test %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  mutate(y_hat_best_preds = mu+b_i+b_u) %>%
  arrange(desc(y_hat_best_preds)) %>%
  select(title) %>%
  unique() %>%
  slice(1:20)
best20_movies

## Best predictions with number of ratings and average rating
bp_lr <- tibble(No. = 1:nrow(best20_movies),
                Movie_Title = character(length = nrow(best20_movies)),
                Count = integer(length = nrow(best20_movies)),
                Average_Rating = numeric(length = nrow(best20_movies)))
# for looping
for (i in seq_along(best20_movies$title)) {
  ind <- which(final_holdout_test$title == as.character(best20_movies$title[i]))
  avg_rating <- mean(final_holdout_test$rating[ind])
  count <- sum(final_holdout_test$title == as.character(best20_movies$title[i]))
  bp_lr$Movie_Title[i] <- best20_movies$title[i]
  bp_lr$Count[i] <- count
  bp_lr$Average_Rating[i] <- round(avg_rating, 2)
}

## Worst movies recommended by the linear model on the final_holdout_test set
worst20_movies <- final_holdout_test %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  mutate(y_hat_best_preds = mu+b_i+b_u) %>%
  arrange(y_hat_best_preds) %>%
  select(title) %>%
  unique() %>%
  slice(1:20)
worst20_movies
```

```r
## Worst predictions with number of ratings and average rating
wp_lr <- tibble(No. = 1:nrow(worst20_movies),
                Movie_Title = character(length = nrow(worst20_movies)),
                Count = integer(length = nrow(worst20_movies)),
                Average_Rating = numeric(length = nrow(worst20_movies)))
# for looping
for (i in seq_along(worst20_movies$title)) {
  ind <- which(final_holdout_test$title == as.character(worst20_movies$title[i]))
  avg_rating <- mean(final_holdout_test$rating[ind])
  count <- sum(final_holdout_test$title == as.character(worst20_movies$title[i]))
  wp_lr$Movie_Title[i] <- worst20_movies$title[i]
  wp_lr$Count[i] <- count
  wp_lr$Average_Rating[i] <- round(avg_rating, 2)
}
```

c. Regularisation linear regression model

For the potential *overfitting* and *underfitting* of the trained adjusted movies and users effects linear regression model leading to a poor performance on external dataset (for e.g., the final hold-out set), a regularisation linear regression system was also modelled[17]. A Ridge regularisation technique was utilised to apply a penalty term in the model training function. The regularisation model is expressed mathematically as:

$$\sum_{u,i} \left(Y_{u,i} - \mu - \beta_i\right)^2 + \lambda \sum_i \beta_i^2$$

The penalty term (λ) that minimises the RMSE was selected using a cross validation approach.

Regularisation function: The function used to train the regularisation linear regression model is given below:

```r
## Generating the regulisation function
regular_fun <- function(lambda, train, test){
  mu <- mean(train_edx$rating)

  movie_bias <- train_edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

  user_bias <- train_edx %>%
    left_join(movie_bias, by="movieId") %>%
    filter(!is.na(b_i)) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - (mu+b_i))/(n()+lambda))

  predicted_ratings <- test_edx %>%
    left_join(movie_bias, by = "movieId") %>%
    left_join(user_bias, by = "userId") %>%
    filter(!is.na(b_i), !is.na(b_u)) %>%
    mutate(pred = mu+b_i+b_u) %>%
```

[17]Irizarry R. A: Introduction to Data Science. Data Analysis and Prediction Algorithms with R. 2024 url: https://rafalab.dfci.harvard.edu/dsbook/

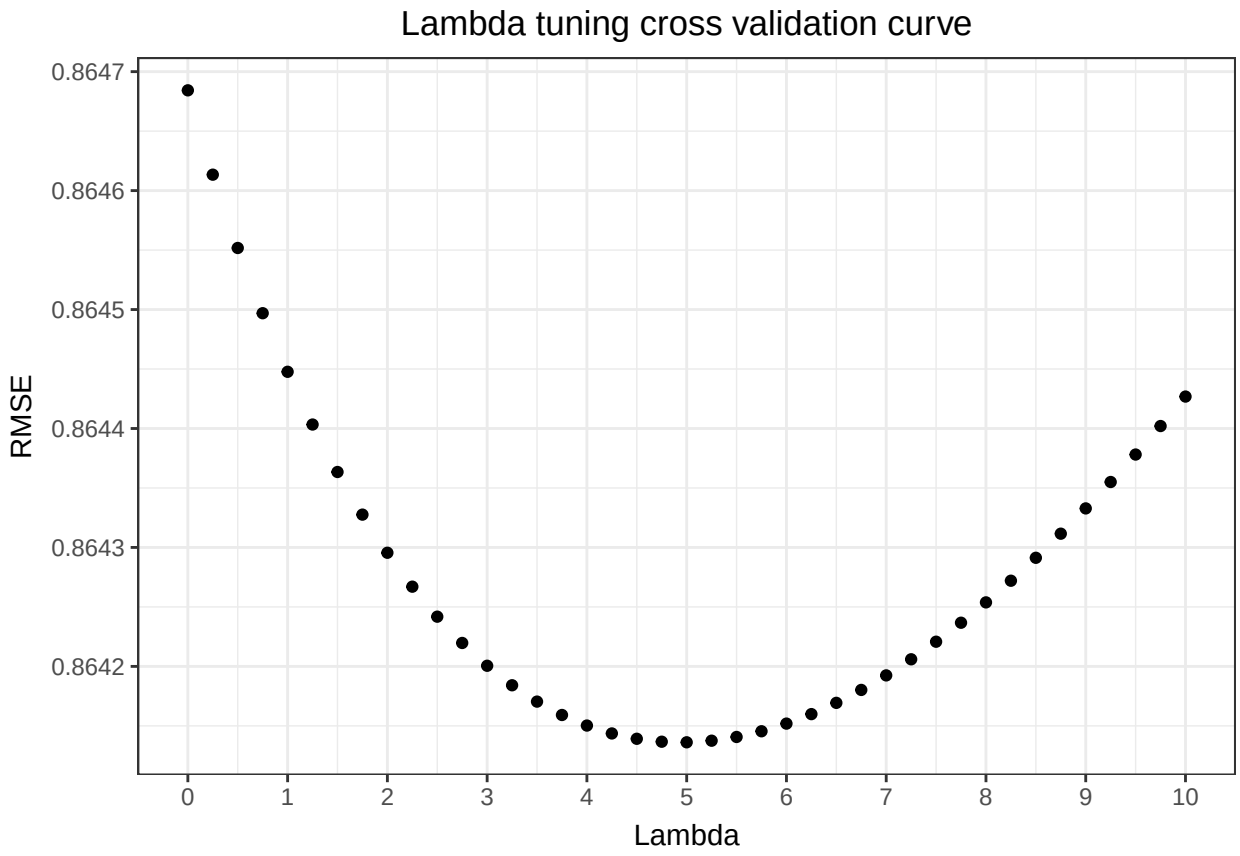
```
    pull(pred)

  return(RMSE(predicted_ratings, test_edx$rating))
}
```

Lambda selection cross validation: The cross-validation method to select the penalty term (λ) was performed as shown below:

```
lambda_rmse <- seq(0, 10, 0.25)
tune_rmse <- sapply(lambda_rmse,
                    regular_fun,
                    train = train_edx,
                    test = test_edx)
```

The lambda tuning cross validation output is visually presented in a line graph.

```
qplot(lambda_rmse, tune_rmse) +
  ggtitle("Lambda tuning cross validation curve") +
   xlab("Lambda") +
  ylab("RMSE") +
  scale_y_continuous(n.breaks = 8) +
  scale_x_continuous(n.breaks = 10) +
  theme_bw() +
  theme(axis.title.x = element_text(vjust = 0),
        axis.title.y = element_text(vjust = 2),
        plot.title = element_text(hjust=0.5))
```



Regularisation system on test_edx set: The trained regularisation linear regression model was first tested on the test_edx set to validate the performance.

```
# Evaluating trained regularisation system on the test_edx data set
lambda <- lambda_rmse[which.min(tune_rmse)]
mu <- mean(train_edx$rating)

movie_bias_regularised <- train_edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

user_bias_regularised <- train_edx %>%
  left_join(movie_bias_regularised, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - (mu+b_i))/(n()+lambda))

y_hat_regularised <- test_edx %>%
  left_join(movie_bias_regularised, by = "movieId") %>%
  left_join(user_bias_regularised, by = "userId") %>%
  mutate(pred_regularised = mu+b_i+b_u) %>%
  pull(pred_regularised)

regularised_rmse <- RMSE(y_hat_regularised, test_edx$rating)
regularised_rmse
```

```
## [1] 0.8641362
```

```
kable(rmse_evaluations <- bind_rows(rmse_evaluations,
                           tibble(Method="Linear regression: regularisation",
                               RMSE = regularised_rmse)))
```

Applying the regularisation on the final hold-out test: Finally, the trained regularisation system was applied to the final hold-out test dataset and the RMSE calculated.

```
### Evaluation with the final_holdout_test set
movie_bias_regularised <- train_edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

user_bias_regularised <- train_edx %>%
  left_join(movie_bias_regularised, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - (mu+b_i))/(n()+lambda))

y_hat_regularised <- final_holdout_test %>%
  left_join(movie_bias_regularised, by = "movieId") %>%
  left_join(user_bias_regularised, by = "userId") %>%
  mutate(pred_regularised = mu+b_i+b_u) %>%
  pull(pred_regularised)

regularised_final_rmse <- RMSE(y_hat_regularised, final_holdout_test$rating)
regularised_final_rmse
```

```
## [1] 0.8652208
```

```
kable(rmse_evaluations <- bind_rows(rmse_evaluations,
              tibble(Method="Linear regression: regularisation on final hold-out test",
                              RMSE = regularised_final_rmse )))
```

The code for movie predictions based on the regularisation system is presented below:

```
### Best movie recommendations by regularisation model
best20_movies_regularised <- final_holdout_test %>%
  left_join(movie_bias_regularised, by = "movieId") %>%
  left_join(user_bias_regularised, by = "userId") %>%
  mutate(y_hat_regularised = mu+b_i+b_u) %>%
  arrange(desc(y_hat_regularised)) %>%
  select(title) %>%
  unique() %>%
  slice(1:20)
best20_movies_regularised

## Best regularisation predictions with number of ratings and average rating
bp_lr_regularised <- tibble(No. = 1:nrow(best20_movies_regularised),
                        Movie_Title = character(length=nrow(best20_movies_regularised)),
                        Count = integer(length=nrow(best20_movies_regularised)),
```

```
                        Average_Rating = numeric(length=nrow(best20_movies_regularised)))
# for looping
for (i in seq_along(best20_movies_regularised$title)) {
  ind<- which(final_holdout_test$title==as.character(best20_movies_regularised$title[i]))
  avg_rating <- mean(final_holdout_test$rating[ind])
  count<- sum(final_holdout_test$title==as.character(best20_movies_regularised$title[i]))
  bp_lr_regularised$Movie_Title[i] <- best20_movies_regularised$title[i]
  bp_lr_regularised$Count[i] <- count
  bp_lr_regularised$Average_Rating[i] <- round(avg_rating, 2)
}

### Worst movie recommendations by regularisation model
worst20_movies_regularised <- final_holdout_test %>%
  left_join(movie_bias_regularised, by = "movieId") %>%
  left_join(user_bias_regularised, by = "userId") %>%
  mutate(y_hat_regularised = mu+b_i+b_u) %>%
  arrange(y_hat_regularised) %>%
  select(title) %>%
  unique() %>%
  slice(1:20)
worst20_movies_regularised

 ## Worst regularisation predictions with number of ratings and average rating
wp_lr_regularised <- tibble(No. = 1:nrow(worst20_movies_regularised),
                        Movie_Title = character(length=nrow(worst20_movies_regularised)),
                        Count = integer(length=nrow(worst20_movies_regularised)),
                        Average_Rating = numeric(length=nrow(worst20_movies_regularised)))
# for looping
for (i in seq_along(worst20_movies_regularised$title)) {
  ind<- which(final_holdout_test$title==as.character(worst20_movies_regularised$title[i]))
  avg_rating <- mean(final_holdout_test$rating[ind])
  count<- sum(final_holdout_test$title==as.character(worst20_movies_regularised$title[i]))
  wp_lr_regularised$Movie_Title[i] <- worst20_movies_regularised$title[i]
  wp_lr_regularised$Count[i] <- count
  wp_lr_regularised$Average_Rating[i] <- round(avg_rating, 2)
}
```

2. Matrix factorisation system

While the linear regression model approach accounts for movies and users variations, groups of movies and users could follow similar rating patterns which were not adequately accounted for in the linear regression model[18]. A matrix factorisation (MF) method with parallel stochastic gradient descent [19] was used to address this potential limitation in the trained linear regression system. The principle underpinning the matrix factorisation is to approximate the entire rating matrix $R_{m\times n}$ by the product of two lower dimensions matrices, $P_{n\times k}$ and $Q_{n\times k}$, such that:

$$R \approx PQ'$$

Let p_u be the u^{th} row of P, and q_v be the v^{th} row of Q, then the rating given by user u on movie i would be predicted as $p_u q_i'$.

[18]Irizarry R. A: Introduction to Data Science. Data Analysis and Prediction Algorithms with R. 2024 URL: https://rafalab.dfci.harvard.edu/dsbook/

[19]Qiu Y. X. : recosystem: Recommender System Using Parallel Matrix Factorization. 2023 URL: https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html

The `recosystem` package was used to train matrices P and Q and the penalty parameters tuning. The `recosystem` is an R wrapper of the `LIBMF` library which creates recommendation systems by using Parallel Matrix Factorization to predict unknown entries in a rating matrix based on observed values[20].

The processes of the MF system training and testing are shown below.

```
## First, the input format edx dataset (both the train_edx and test_edx) is transformed
## into recosystem format
set.seed(1)
recosys_train <- with(train_edx, data_memory(user_index = userId,
                                             item_index = movieId, rating = rating))
recosys_test <- with(test_edx, data_memory(user_index = userId,
                                           item_index = movieId, rating = rating))

## Creating the model object
recosys <- Reco()

## Modeltuning
model_tune <- recosys$tune(recosys_train, opts = list(dim = c(10, 20, 30),
                                                      lrate = c(0.1, 0.2), nthread  = 4,
                                                      niter = 10))

## training the model
recosys$train(recosys_train, opts = c(model_tune$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9818   1.1025e+07
##    1       0.8751   8.9894e+06
##    2       0.8426   8.3451e+06
##    3       0.8201   7.9542e+06
##    4       0.8036   7.6848e+06
##    5       0.7917   7.4943e+06
##    6       0.7818   7.3554e+06
##    7       0.7733   7.2377e+06
##    8       0.7660   7.1463e+06
##    9       0.7596   7.0657e+06
##   10       0.7539   7.0010e+06
##   11       0.7488   6.9427e+06
##   12       0.7441   6.8917e+06
##   13       0.7400   6.8464e+06
##   14       0.7362   6.8077e+06
##   15       0.7328   6.7727e+06
##   16       0.7295   6.7404e+06
##   17       0.7266   6.7136e+06
##   18       0.7239   6.6895e+06
##   19       0.7214   6.6661e+06
```

The performance of the trained MF system was first validated on the test_edx set.

[20]Chin, Wei-Sheng, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin: A Fast Parallel Stochastic Gradient Method for Matrix Factorization in Shared Memory Systems. ACM TIST. 2015a. URL: https://www.csie.ntu.edu.tw/~cjlin/papers/libmf/libmf_journal.pdf.

```
## Making prediction on the recosys_test (test_edx) set
y_hat_mf <-  recosys$predict(recosys_test, out_memory()) ## Return an R vector

mf_rmse_testedx <- RMSE(y_hat_mf, test_edx$rating)
mf_rmse_testedx
```

```
## [1] 0.7855584
```

```
kable(rmse_evaluations <- bind_rows(rmse_evaluations,
                              tibble(Method="Matrix factorisation system",
                                     RMSE = mf_rmse_testedx )))
```

Further, the trained MF was applied on the final hold out test data set.

```
## Movie recommendations by the matrix factorization model
recosys_finaltest <- with(final_holdout_test, data_memory(user_index = userId,
                                                          item_index = movieId, rating = rating))

## prediction on final holdout test
y_hat_mf_finaltest <-  recosys$predict(recosys_finaltest, out_memory())

mf_rmse_finaltest <- RMSE(y_hat_mf_finaltest, final_holdout_test$rating)
mf_rmse_finaltest
```

```
## [1] 0.7859912
```

```
kable(rmse_evaluations <- bind_rows(rmse_evaluations,
                      tibble(Method="Matrix factorisation system on final hold-out test",
                                     RMSE = mf_rmse_finaltest )))
```

Furthermore, movie predictions by the trained MF system were performed as follow:

```
 #  Best recommendations
 best20_movies_mf <- data.frame(title=final_holdout_test$title,
                                pred_rating=y_hat_mf_finaltest) %>%
   arrange(desc(pred_rating)) %>%
   select(title) %>%
  unique() %>%
  slice(1:20)
 best20_movies_mf

 # Worst recommendations
 worst20_movies_mf <- data.frame(title=final_holdout_test$title,
                                 pred_rating=y_hat_mf_finaltest) %>%
   arrange(pred_rating) %>%
   select(title) %>%
  unique() %>%
  slice(1:20)
worst20_movies_mf

## Best MF predictions with number of ratings and average rating
```

```r
bp_mf <- tibble(No. = 1:nrow(best20_movies_mf),
                Movie_Title = character(length = nrow(best20_movies_mf)),
                Count = integer(length = nrow(best20_movies_mf)),
                Average_Rating = numeric(length = nrow(best20_movies_mf)))
# for looping
for (i in seq_along(best20_movies_mf$title)) {
  ind <- which(final_holdout_test$title == as.character(best20_movies_mf$title[i]))
  avg_rating <- mean(final_holdout_test$rating[ind])
  count <- sum(final_holdout_test$title == as.character(best20_movies_mf$title[i]))
  bp_mf$Movie_Title[i] <- best20_movies_mf$title[i]
  bp_mf$Count[i] <- count
  bp_mf$Average_Rating[i] <- round(avg_rating, 2)
}

# Worst recommendations - MF
worst20_movies_mf <- data.frame(title=final_holdout_test$title,
                                pred_rating=y_hat_mf_finaltest) %>%
  arrange(pred_rating) %>%
  select(title) %>%
  unique() %>%
  slice(1:20)
worst20_movies_mf

## Worst MF predictions with number of ratings and average rating
wp_mf <- tibble(No. = 1:nrow(worst20_movies_mf),
                Movie_Title = character(length = nrow(worst20_movies_mf)),
                Count = integer(length = nrow(worst20_movies_mf)),
                Average_Rating = numeric(length = nrow(worst20_movies_mf)))
# for looping
for (i in seq_along(worst20_movies_mf$title)) {
  ind <- which(final_holdout_test$title == as.character(worst20_movies_mf$title[i]))
  avg_rating <- mean(final_holdout_test$rating[ind])
  count <- sum(final_holdout_test$title == as.character(worst20_movies_mf$title[i]))
  wp_mf$Movie_Title[i] <- worst20_movies_mf$title[i]
  wp_mf$Count[i] <- count
  wp_mf$Average_Rating[i] <- round(avg_rating, 2)
}
```

Results

1. **Movie predictions on the final hold-out dataset**

The trained recommendation systems were applied to make movie predictions on the final hold-out test data set. The tables present the best and worst predicted movies, the total number of ratings, and the average ratings per movie of the trained recommendation systems.

a. ***Best movie predictions***

The top 20 movies predicted by the recommendation systems are presented below.

i. **Linear regression**

This table shows the best movies predicted by the adjusted movies' and users' effects linear regression model.

```
knitr::kable(bp_lr, caption = "Best linear regression system predicted movies")
```

Table 1: Best linear regression system predicted movies

No.	Movie_Title	Count	Average_Rating
1	Usual Suspects, The (1995)	2389	4.38
2	Shawshank Redemption, The (1994)	3111	4.48
3	Schindler's List (1993)	2584	4.36
4	Eternal Sunshine of the Spotless Mind (2004)	859	4.18
5	Wallace & Gromit: A Close Shave (1995)	642	4.27
6	Star Wars: Episode VI - Return of the Jedi (1983)	2514	4.00
7	Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)	2125	4.28
8	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	2894	4.21
9	Fargo (1996)	2399	4.12
10	Godfather, The (1972)	2067	4.41
11	Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966)	704	4.11
12	Pulp Fiction (1994)	3502	4.18
13	Donnie Darko (2001)	718	4.09
14	Saving Private Ryan (1998)	1821	4.07
15	Shining, The (1980)	1193	4.00
16	Boat, The (Das Boot) (1981)	733	4.16
17	Silence of the Lambs, The (1991)	3286	4.21
18	Beautiful Thing (1996)	81	3.91
19	Terminator 2: Judgment Day (1991)	2964	3.93
20	Princess Bride, The (1987)	1711	4.18

ii. **Regularisation linear regression**

The best predicted movies based on the regularisation linear regression system are presented in the table below.

```
knitr::kable(bp_lr_regularised, caption = "Best regularisation system predicted movies")
```

Table 2: Best regularisation system predicted movies

No.	Movie_Title	Count	Average_Rating
1	Usual Suspects, The (1995)	2389	4.38
2	Shawshank Redemption, The (1994)	3111	4.48
3	Eternal Sunshine of the Spotless Mind (2004)	859	4.18
4	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	2894	4.21
5	Star Wars: Episode VI - Return of the Jedi (1983)	2514	4.00
6	Schindler's List (1993)	2584	4.36
7	Donnie Darko (2001)	718	4.09
8	Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)	2125	4.28

No.	Movie_Title	Count	Average_Rating
9	Shining, The (1980)	1193	4.00
10	Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966)	704	4.11
11	Saving Private Ryan (1998)	1821	4.07
12	Boat, The (Das Boot) (1981)	733	4.16
13	Pulp Fiction (1994)	3502	4.18
14	Godfather, The (1972)	2067	4.41
15	Kill Bill: Vol. 1 (2003)	844	3.90
16	Hoop Dreams (1994)	681	4.05
17	Forrest Gump (1994)	3378	4.02
18	Princess Bride, The (1987)	1711	4.18
19	Beautiful Thing (1996)	81	3.91
20	Big Night (1996)	341	4.01

iii. **Matrix factorisation**

The MF system best predicted movies are shown in the table below.

```
knitr::kable(bp_mf, caption = "Best matrix factorisation system predicted movies")
```

Table 3: Best matrix factorisation system predicted movies

No.	Movie_Title	Count	Average_Rating
1	Lord of the Rings: The Return of the King, The (2003)	1253	4.16
2	Shawshank Redemption, The (1994)	3111	4.48
3	Godfather, The (1972)	2067	4.41
4	Annie Hall (1977)	818	4.12
5	Rhyme & Reason (1997)	2	4.00
6	Boys Life 2 (1997)	5	4.60
7	Schindler's List (1993)	2584	4.36
8	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	2894	4.21
9	Star Wars: Episode V - The Empire Strikes Back (1980)	2362	4.21
10	Alien (1979)	1590	4.04
11	Fight Club (1999)	1602	4.19
12	South Park: Bigger, Longer and Uncut (1999)	1009	3.59
13	Monty Python and the Holy Grail (1975)	1607	4.21
14	Matrix, The (1999)	2321	4.23
15	Braveheart (1995)	2942	4.09
16	Usual Suspects, The (1995)	2389	4.38
17	Cats Don't Dance (1997)	30	2.95
18	Manhattan (1979)	442	4.06
19	Princess Bride, The (1987)	1711	4.18
20	Happy Together (1989)	4	2.88

b. ***Worst movie predictions***

The bottom 20 movies predicted by the recommendation systems are presented in the tables below.

i. **Linear regression**

The table below shows the worst movies predicted by the adjusted movies' and users' effects linear regression system.

```
knitr::kable(wp_lr, caption = "Worst linear regression system predicted movies")
```

Table 4: Worst linear regression system predicted movies

No.	Movie_Title	Count	Average_Rating
1	Battlefield Earth (2000)	203	1.67
2	Tar (1996)	3	1.67
3	Police Academy 4: Citizens on Patrol (1987)	131	2.06
4	Karate Kid Part III, The (1989)	211	2.06
5	Pokémon Heroes (2003)	19	0.87
6	Turbo: A Power Rangers Movie (1997)	46	1.37
7	Problem Child (1990)	84	1.97
8	Kazaam (1996)	111	1.84
9	Shanghai Surprise (1986)	10	1.15
10	Faces of Death 6 (1996)	15	1.17
11	Camp Rock (2008)	2	1.25
12	Carnosaur 3: Primal Species (1996)	11	1.32
13	Free Willy 3: The Rescue (1997)	47	1.86
14	Iron Eagle IV (1995)	34	1.63
15	Barney's Great Adventure (1998)	26	1.08
16	War of the Worlds 2: The Next Wave (2008)	1	1.00
17	RoboCop 2 (1990)	270	2.52
18	Steel (1997)	30	1.43
19	House of the Dead, The (2003)	22	1.09
20	Super Mario Bros. (1993)	302	2.06

ii. **Regularisation linear regression**

The worst predicted movies based on the regularisation linear regression system are presented in the table below.

```
knitr::kable(wp_lr_regularised, caption = "Worst regularisation system predicted movies")
```

Table 5: Worst regularisation system predicted movies

No.	Movie_Title	Count	Average_Rating
1	Battlefield Earth (2000)	203	1.67
2	Police Academy 4: Citizens on Patrol (1987)	131	2.06
3	Karate Kid Part III, The (1989)	211	2.06
4	Pokémon Heroes (2003)	19	0.87
5	Kazaam (1996)	111	1.84
6	Turbo: A Power Rangers Movie (1997)	46	1.37
7	Shanghai Surprise (1986)	10	1.15
8	Free Willy 3: The Rescue (1997)	47	1.86
9	RoboCop 2 (1990)	270	2.52
10	Iron Eagle IV (1995)	34	1.63
11	Faces of Death 6 (1996)	15	1.17

No.	Movie_Title	Count	Average_Rating
12	House of the Dead, The (2003)	22	1.09
13	Barney's Great Adventure (1998)	26	1.08
14	Steel (1997)	30	1.43
15	Aces: Iron Eagle III (1992)	49	1.76
16	Super Mario Bros. (1993)	302	2.06
17	Alone in the Dark (2005)	20	1.70
18	Spice World (1997)	150	1.78
19	Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)	31	1.16
20	Glitter (2001)	41	1.10

iii. **Matrix factorisation**

The worst predicted movies by the MF system are shown in the table below.

```
knitr::kable(wp_mf, caption = "Worst matrix factorisation system predicted movies")
```

Table 6: Worst matrix factorisation system predicted movies

No.	Movie_Title	Count	Average_Rating
1	Dead Girl, The (2006)	3	2.83
2	Beast of Yucca Flats, The (1961)	2	0.75
3	Alien from L.A. (1988)	4	1.62
4	Blue Kite, The (Lan feng zheng) (1993)	4	2.25
5	What the #$*! Do We Know!? (a.k.a. What the Bleep Do We Know!?) (2004)	27	2.78
6	What Time Is It There? (Ni neibian jidian) (2001)	11	3.45
7	Daddy Day Camp (2007)	2	1.25
8	Free Willy 2: The Adventure Home (1995)	185	2.05
9	Time Walker (a.k.a. Being From Another Planet) (1982)	1	0.50
10	3 Ninjas Knuckle Up (1995)	24	1.44
11	Texas Chainsaw Massacre, The (1974)	150	3.25
12	Ernest in the Army (1998)	14	1.96
13	Up the Academy (1980)	1	1.00
14	Big Nothing (2006)	5	3.30
15	Texas Chainsaw Massacre 2, The (1986)	39	2.19
16	Canterbury Tale, A (1944)	3	3.00
17	Nightcap (Merci Pour le Chocolat) (2000)	3	2.67
18	Armageddon (1998)	1108	2.92
19	Pokémon Heroes (2003)	19	0.87
20	Freddy Got Fingered (2001)	82	1.93

2. **The quality of the recommendation systems**

The calculated RMSE of all the models iterated and predictions on the final hold-out test are presented in the table below.

```
knitr::kable(rmse_evaluations, caption = "Recommendation systems evaluation") %>%
  kable_classic(full_width = F)
```

Table 7: Recommendation systems evaluation

Method	RMSE
Linear regression: base model	1.0600537
Linear regression: with movies' effects	0.9429615
Linear regression: adjusted movies' & users' effects	0.8646843
Linear regression: movies' & users' effects on final hold-out test	0.8658528
Linear regression: regularisation	0.8641362
Linear regression: regularisation on final hold-out test	0.8652208
Matrix factorisation system	0.7855584
Matrix factorisation system on final hold-out test	0.7859912

Conclusion

This capstone report on MovieLens 10M dataset movie recommendation systems demonstrates the knowledge acquired in the HarvardX Data Science profession certificate course, specifically the competence attained in basics of machine learning recommendation systems. The MovieLens dataset was downloaded, cleaned and partitioned into a training set to build the algorithms utilising a linear regression (with regularisation) and matrix factorisation methods, and a testing set for evaluating the quality of the built recommendation systems.

After training the recommendation system algorithms, the performance were checked on a 'final holdout test' dataset, and the best and worst movie predictions were conducted using this (final holdout test) dataset. The observed *quality* of the trained system based on linear regression with adjusted movies' and users' effect ($RMSE = 0.866$) was similar to the linear regression with regularisation trained model ($RMSE = 0.865$), with the regularisation system performing slightly better. Unsurprisingly, there were similarities in the movies predicted based on these linear regression-trained recommendation systems. In contrast, the MF recommendation system performed much better ($RMSE = 0.786$) than the linear regression recommendation systems. There were also substantial variations in the movies predicted by the trained matrix factorisation system on the final hold-out test dataset. The observed differences in the *quality* of the recommendation systems could be explained partly by the dataset's structure with the matrix factorisation method being more flexible to handle the complexity in the data to minimise overfitted trained system; therefore, more generalisable on external data sets.

It is noteworthy that the built algorithms were based on only two predictor variables (features: movies and users); however, the MovieLens dataset has several parameters (variables), including genres, time of rating, and movie's released year that can be important attributes of movie rating but were not considered in building the recommendation systems. Other potential recommendation systems could also consider accounting for these additional variables in the model training to see the performance of the predictions. Also, it is worth mentioning that further penalty parameters tuning could improve the recommendation systems, however, the process could become computationally heavy and crash the R system[21].

Finally, this capstone project on movie recommendation systems provided an opportunity to apply and further enhanced the knowledge and skills acquired in the Data Science course.

Acknowledgment

The course textbook and other online resources some of which are cited in this document have been helpful throughout the course.

[21] Outerelo C.: MovieLens Recommender System. (undated) assessed on June 2025. URL: https://rpubs.com/outerelocarlos/MovieLens-Recommender-System