HarvardX PH125.9x: Movielens project

Yin-Chi Chan

2022-04-10

Contents

Preface

This book is available in both HTML gitbook and PDF form.

The source code in the PDF version of this report is typeset in Cascadia Code, with code ligatures enabled.

0.1 R setup

```
# R 4.1 key features: new pipe operator, \(x) as shortcut for function(x)
# R 4.0 key features: stringsAsFactors = FALSE by default, raw character strings r"()"
if (packageVersion('base') < '4.1.0') {
  stop('This code requires R ≥ 4.1.0!')
```

```
}

if(!require("pacman")) install.packages("pacman")
library(pacman)

# Ensure packages are installed but do not load them
p_install(Rcpp, force = F)
p_install(RcppArmadillo, force = F)
p_install(RcppProgress, force = F)
p_install(mgcv, force = F)          # provides mgcv::gam and mgcv::predict.gam
p_install(raster, force = F)        # provides raster::clamp

# Load these packages
p_load(magrittr, knitr, kableExtra, data.table, latex2exp, patchwork,
       tidyverse, caret, lubridate)
```

1 Introduction

This report partially fulfills the requirements for the HarvardX course PH125.9x: "Data Science: Capstone". The objective of this process is to build a movie recommendation system using the MovieLens dataset. The 10M version (GroupLens 2009) of this dataset was used for this project.

Using the code provided by the course, the 10 million records of the MovieLens 10M dataset are split into the `edx` partition, for building the movie recommendation system, and the `validation` partition, for evaluating the proposed system. The `validation` dataset contains roughly 10 percent of the records in the MovieLens 10M dataset. The code for generating these two datasets is provided below:

```
##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

# NOTE: this code was modified from the course-provided version for speed.

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


# STEP 1: download and unzip 'ml-10m.zip' as necessary

if(!dir.exists("ml-10M100K")) dir.create("ml-10M100K")
dl <- "ml-10M100K/ratings.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)


# STEP 2a: Load the ratings file.  This file is delimited using double colons.

ratings <- str_split(read_lines(ratings_file), fixed("::"), simplify = T) |>
  as.data.frame() |>
  set_colnames(c("userId", "movieId", "rating", "timestamp")) |>
  mutate(userId = as.integer(userId),
```

```r
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as_datetime(as.integer(timestamp)))


# STEP 2b: Load the movies file.  Again, this file is delimited using double colons.
movies <- str_split(read_lines(movies_file), fixed("::"), simplify = T) |>
  as.data.frame() |>
  set_colnames(c("movieId", "title", "genres")) |>
  mutate(movieId = as.integer(movieId))


# STEP 3: Join the `ratings` and `movies` data tables and save to `movielens`.
movielens <- left_join(ratings, movies, by = "movieId")


# STEP 4: Split the `movielens` dataset into the `edx` and `validation` sets.

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
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 validation set are also in edx set
validation <- temp |> semi_join(edx, by = "movieId") |> semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)


# STEP 5; convert timestamps to datetime
edx <- edx |> mutate(timestamp = as_datetime(timestamp)) |> as.data.table()
validation <- validation |> mutate(timestamp = as_datetime(timestamp)) |> as.data.table()

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

1.1 Data description

The data consists of ten million movie ratings, each expressed using six variables. The number of ratings in the `edx` and `validation` partitions are `nrow(edx)` and `nrow(validation)`, respectively, i.e., 9000055 and 999999. The six variables are:

```r
colnames(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

and are defined as follows:

- `userId`: an integer from 1 to 71567 denoting the user who made the rating.
- `movieId`: an integer from 1 to 65133 denoting which movie was rated.
- `rating`: a multiple of 0.5, from 0.5 to 5.0.
- `timestamp`: a `POSIXct` object representing the time at which the rating was made.
- `title`: the name of the movie rated, suffixed which the year of release in parentheses.
- `genres`: a list of genres for the rated movie, delimited by the pipe ('|') character.

Note that only integer ratings were supported before February 2003; the earliest half-star rating is:

```r
temp <- edx[edx$rating %% 1 == 0.5]
temp[which.min(temp$timestamp)] |> kable(align='rrrrll', booktabs = T) |> row_spec(0, bold = T)
```

userId	movieId	rating	timestamp	title	genres
53996	4018	3.5	2003-02-12 17:31:34	What Women Want (2000)	Comedy|Romance

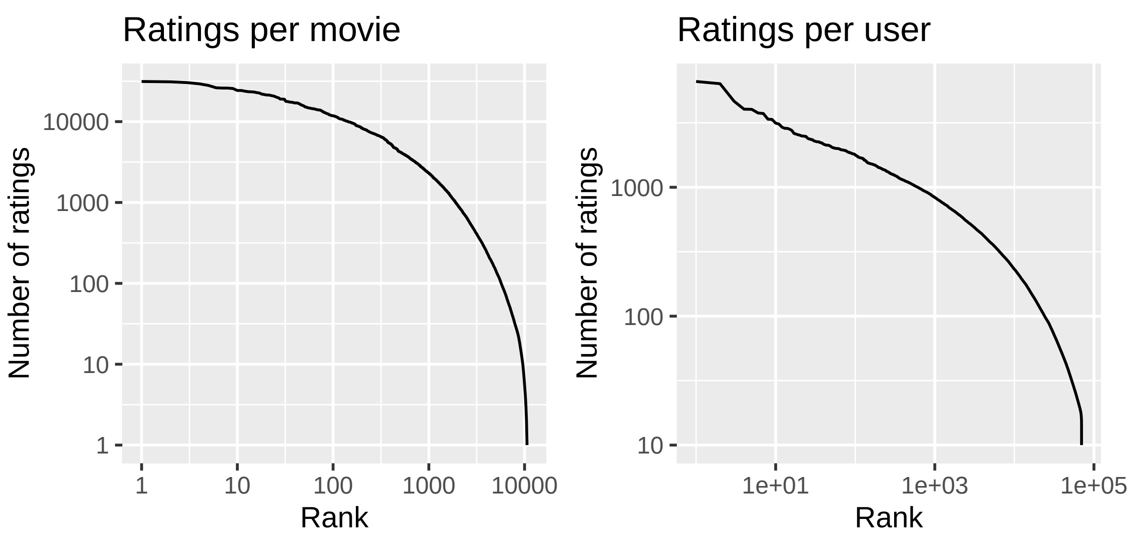
The density of the rating matrix is:

```r
nrow(edx) / max(edx$userId) / max(edx$movieId)
```

```
## [1] 0.001930773
```

The number of ratings per movie and per user in `edx` is plotted below.

```r
# Count the number of ratings for each movie and rank the movies by ratings received
ratings_per_movie <- edx |>
  group_by(movieId) |>
  summarise(title = first(title), genres = first(genres), n_ratings = n()) |>
  mutate(rank = frank(-n_ratings))

# Count the number of ratings for each user and rank the users by ratings given
ratings_per_user <- edx |>
  group_by(userId) |>
  summarise(n_ratings = n()) |>
  mutate(rank = frank(-n_ratings))

# Plot the number of ratings for each movie and user, sorted by rank
plot1 <- ratings_per_movie |>
  ggplot(aes(rank,n_ratings)) + geom_line() +
  scale_x_log10() + scale_y_log10() +
  xlab('Rank') + ylab('Number of ratings') + labs(title = 'Ratings per movie')
plot2 <- ratings_per_user |>
  ggplot(aes(rank,n_ratings)) + geom_line() +
  scale_x_log10() + scale_y_log10() +
  xlab('Rank') + ylab('Number of ratings') + labs(title = 'Ratings per user')

rm(ratings_per_movie, ratings_per_user)
par(cex = 0.7)
plot1 + plot2
```



1.1.1 Movie genres

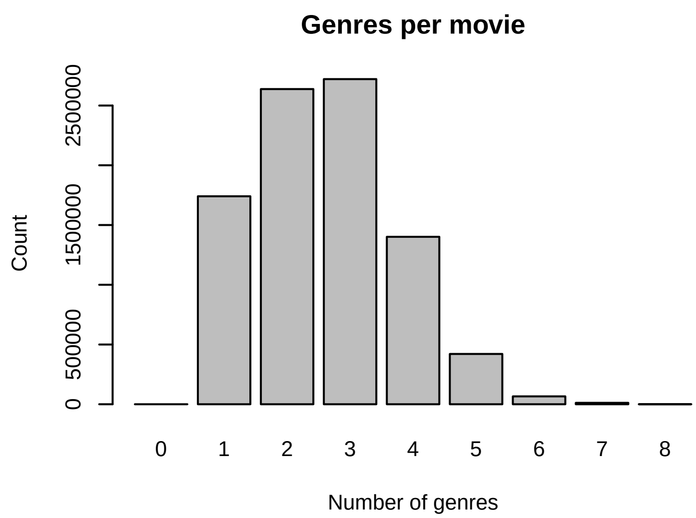
The list of possible genres, the number of movies in each genre, and the mean number of ratings per movie in each genre is given as follows:

```
# Get the list of possible genres

genres <- edx$genres |> unique() |> str_split('\\|') |> flatten_chr() |>
  unique() |> sort() |>
  tail(-1) # remove "(no genres listed)"

# Construct a data.table with one entry per movie
temp <- edx |> group_by(movieId) |>
  summarise(title = first(title), genres = first(genres))

# Find the number of movies and ratings for each genre
genre_summary <-
  data.table(
    Genre = genres,
    Movies = sapply(genres, function(g)
      sum(temp$genres %flike% g)),
    Ratings = sapply(genres, function(g)
      sum(edx$genres %flike% g)),
    "Mean Rating" = sapply(genres, function(g) {
      edx[edx$genres %flike% g,'rating']$rating |> mean()
    })
  ) |>
  mutate("Ratings per movie" = Ratings / Movies)

rm(temp)
genre_summary |> arrange(desc(`Mean Rating`)) |>
  kable(align='lrrrr', digits = c(0,0,0,2,1), booktabs = T, linesep = "") |> row_spec(0, bold = T)
```

Genre	**Movies**	**Ratings**	**Mean Rating**	**Ratings per movie**
Film-Noir	148	118541	4.01	801.0
Documentary	481	93066	3.78	193.5
War	510	511147	3.78	1002.2
IMAX	29	8181	3.77	282.1
Mystery	509	568332	3.68	1116.6
Drama	5336	3910127	3.67	732.8
Crime	1117	1327715	3.67	1188.6
Animation	286	467168	3.60	1633.5
Musical	436	433080	3.56	993.3
Western	275	189394	3.56	688.7
Romance	1685	1712100	3.55	1016.1
Thriller	1705	2325899	3.51	1364.2
Fantasy	543	925637	3.50	1704.7
Adventure	1025	1908892	3.49	1862.3
Comedy	3703	3540930	3.44	956.2
Action	1473	2560545	3.42	1738.3
Children	528	737994	3.42	1397.7
Sci-Fi	754	1341183	3.40	1778.8
Horror	1013	691485	3.27	682.6

The number of genres for each movie is plotted as a histogram below:

```
genre_counts <- table(str_count(edx$genres, '\\|') + 1 - str_count(edx$genres, 'no genres'))
par(cex = 0.7)
barplot(genre_counts, xlab = 'Number of genres', ylab = 'Count', main = 'Genres per movie')
```



Therefore it is likely better to analyze genre combinations rather than individual genres. The following code confirms that over half of all movies have either two or three genres:

```
sum(genre_counts[c('2','3')])/sum(genre_counts)
```

```
## [1] 0.595434
```

1.2 Project Objective

The objective of this project is to estimate movie ratings given the values of the other five variables. The goodness of the proposed recommender system is evaluated using the root mean squared error (RMSE):

$$\mathrm{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} \left(y_{u,i} - \hat{y}_{u,i}\right)^2}$$

where y denotes the true values of movie ratings in the test set $\mathcal{T}$, $\hat{y}$ denotes the estimated values, and N denotes the number of observations in the test set.

The library function `caret::RSME` is used in this report for RSME evaluation.

Note that minimizing the RMSE is equivalent to minimizing the sum of the square errors, i.e.,

$$\mathrm{SE} = \sum_{(u,i)\in\mathcal{T}} \left(y_{u,i} - \hat{y}_{u,i}\right)^2.$$

In matrix form, this can be thought of as the square of the $L_{2,2}$ or Frobenius norm of the prediction errors, i.e.,

$$\mathrm{SE} = \left\|Y - \hat{Y}\right\|_{2,2}^2,$$

where $Y - \hat{Y}$ is defined as zero for user-movie pairs not in the test set.

2 Linear regression models

We start by splitting `edx` into a training and test set:

```
# Test set will be 10% of edx data
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
```

```r
edx_train <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in edx_test set are also in edx_train set
edx_test <- temp |>
  semi_join(edx_train, by = "movieId") |>
  semi_join(edx_train, by = "userId") |>
  as.data.table()

# Add rows removed from edx_test set back into edx_train set
removed2 <- anti_join(temp, edx_test)
edx_train <- rbind(edx_train, removed2) |> as.data.table()

rm(removed2, temp, test_index)
```

2.1 Overview and notation

Let Y be a $N_{\mathrm{U}} \times N_{\mathrm{M}}$ matrix of movie ratings, such that $Y_{u,i}$ is the rating user u has given or would give movie i. Additionally, define X_j such that $X_{u,i}$ denotes the jth attribute of user-movie pair (u,i). Such attributes include u and i themselves, the genres of movie i, and the timestamp at which the rating was made. Finally, only the indices (u,i) in the training set, denoted $\mathcal{T}$, are observable.

The goal is to estimate Y given the observable elements of Y (the actual ratings). Given a user-movie pair (u,i), we model Y_r using a multiple linear regression model:

$$y_{u,i} \sim \mu + \left(\sum_j \beta_{j;u,i}\right) + \varepsilon_{u,i},$$

where

- μ represents the "true" rating for all movies,
- $\beta_j; u, i$ is the jth bias term for pair (u,i),
- and $\varepsilon_{u,i}$ is random error, all independently sampled from the same zero-mean distribution.

We further define b_j such that

$$\left(X_{j;u,i} = n\right) \implies \left(\beta_{j;u,i} = b_{j;n}\right).$$

We can write the above in matrix form:

$$Y \sim \mu + \left(\sum_j \beta_j\right) + \varepsilon.$$

The objective is to minimize the sum of the squared errors

$$\mathrm{SE} = \sum_{(u,i)\in\mathcal{T}} \left[Y_{u,i} - \mu - \sum_j \beta_{j;u,i}\right]^2$$

where $\mathcal{T}$ represents the test set of observed movie ratings.

The estimated value of $Y_{u,v}$ for $(u,v) \notin \mathcal{T}$ is

$$\hat{Y}_{u,v} = \mu + \sum_j \beta_{j;u,v}.$$

2.2 Using the mean rating only

Our first model is of the form

$$Y_{u,i} \sim \mu + \varepsilon_{u,i}.$$

The best estimate $\hat{\mu}$ of μ is the mean of all ratings in `edx_train`, or:

```
mu ← mean(edx_test$rating)
mu
```

```
## [1] 3.512551
```

This model gives the following RMSE values when applied to `edx_test`:

```
# When multiple effects (movie, user, genre) are added in our model, some predictions
# may fall out of the valid range.  This function fixes these predictions to the range
# [0.5, 5].
clamp ← function(x) raster::clamp(as.numeric(x), 0.5, 5)

# Compute RMSE and add to a tibble.
RMSEs ← tibble(Method = c("Mean only"),
                RMSE = RMSE(mu, edx_test$rating),
                "RMSE (clamped estimates)" = RMSE(mu, edx_test$rating))
RMSEs[[nrow(RMSEs),'RMSE']]
```

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

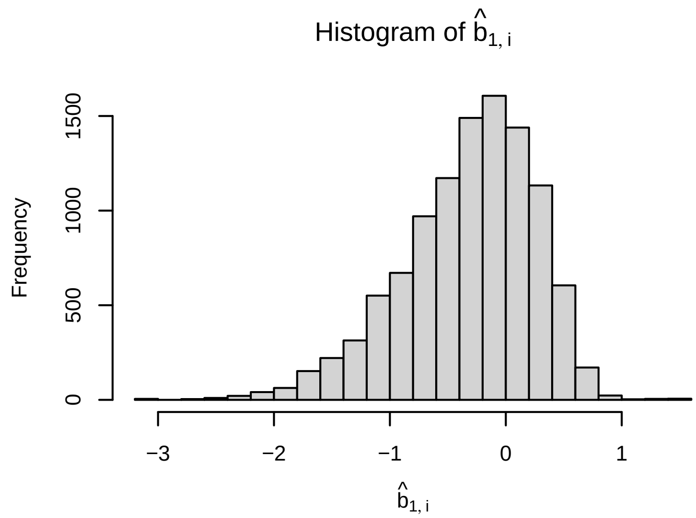
2.3 Modeling movie effects

We add a term to our model for movie effects:

$$Y_{u,i} \sim \mu + b_{1;i}i + \varepsilon_{u,i},$$

The least-squares estimate $\hat{b}_{1;i}$ of $b_{1;i}$ is the training-set mean of $Y_{u,i} - \hat{\mu}$ for each movie i. The following code computes $\hat{b}_{1;i}$ for each i and plots these as a histogram:

```
# Least-squares estimate of movie effect is the mean of (rating - mu) for all
# ratings of that movie.
movie_biases ← edx_train ▷
  group_by(movieId) ▷
  summarize(b_i = mean(rating - mu))

# Plot a histogram of the movie effects
par(cex = 0.7)
hist(movie_biases$b_i, 30, xlab = TeX(r'[$\hat{b}_{1,i}$]'),
     main = TeX(r'[Histogram of $\hat{b}_{1,i}$]'))
```



The new model gives the following RMSE values when applied to `edx_test`:

```
# Obtain predictions for the edx_test set
predicted_ratings <- edx_test |>
  left_join(movie_biases, by='movieId') |>
  mutate(pred = mu + b_i) |> pull(pred)

# Compute RMSE and add to data.table
RMSEs <- RMSEs |>
  add_row(Method = "Movie effects",
          RMSE = RMSE(predicted_ratings, edx_test$rating),
          "RMSE (clamped estimates)" = RMSE(clamp(predicted_ratings), edx_test$rating))

RMSEs[nrow(RMSEs),] |> kable(align='lrr', booktabs = T) |> row_spec(0, bold = T)
```

Method	**RMSE**	**RMSE (clamped estimates)**
Movie effects	0.9429615	0.9429615

2.3.1 Clamping the predictions

In the above table, clamping means setting any predictions less than 0.5 to 0.5, and any predictions greater than 5.0 to 5.0, thus enforcing the limits of possible ratings. This slightly reduces the RMSE when multiple biases are added to the model, as we demonstrate below.

2.4 Modeling movie and user effects

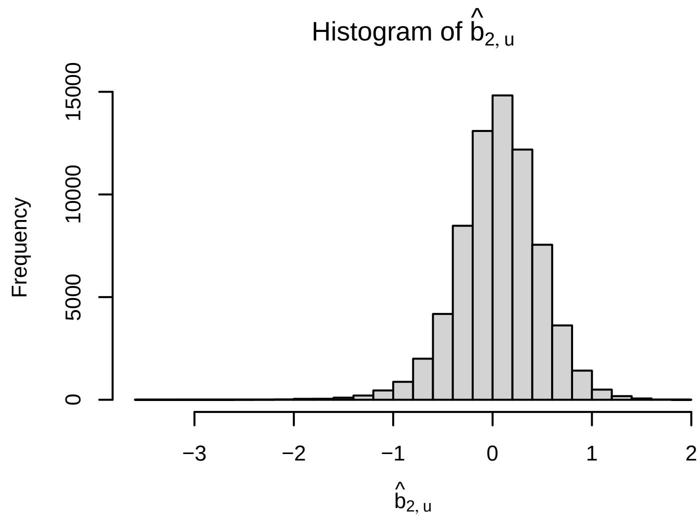
We add a term b_u to our model for user effects:

$$Y_{u,i} \sim \mu + b_{1;i}i + b_{2;u}u + \varepsilon_{u,i}.$$

We approximate $b_{2,u}$ for each user u as the mean of $\hat{b}_u = Y_{u,i} - \hat{\mu} - \hat{b}_{1;i}$. The following code computes $\hat{b}_{2;u}$ for each u and plots these as a histogram:

```
# Estimate user effects
user_biases <- edx_train |>
  left_join(movie_biases, by='movieId') |>
  group_by(userId) |>
  summarize(b_u = mean(rating - mu - b_i))
```

```r
# Plot a histogram of the user effects
par(cex = 0.7)
hist(user_biases$b_u, 30, xlab = TeX(r'[$\hat{b}_{2,u}$]'),
     main = TeX(r'[Histogram of $\hat{b}_{2,u}$]'))
```



The new model gives the following RMSE values when applied to the `edx_test` set:

```r
# Obtain predictions for the edx_test set
predicted_ratings <- edx_test |>
  left_join(movie_biases, by='movieId') |>
  left_join(user_biases, by='userId') |>
  mutate(pred = mu + b_i + b_u) |> pull(pred)

# Compute RMSE and add to data.table
RMSEs <- RMSEs |>
  add_row(Method = "Movie + user effects",
          RMSE = RMSE(predicted_ratings, edx_test$rating),
          "RMSE (clamped estimates)" = RMSE(clamp(predicted_ratings), edx_test$rating))
RMSEs[nrow(RMSEs),] |> kable(align='lrr', booktabs = T) |> row_spec(0, bold = T)
```

Method	**RMSE**	**RMSE (clamped estimates)**
Movie + user effects	0.8646843	0.8644818

2.5 Adding genre effects

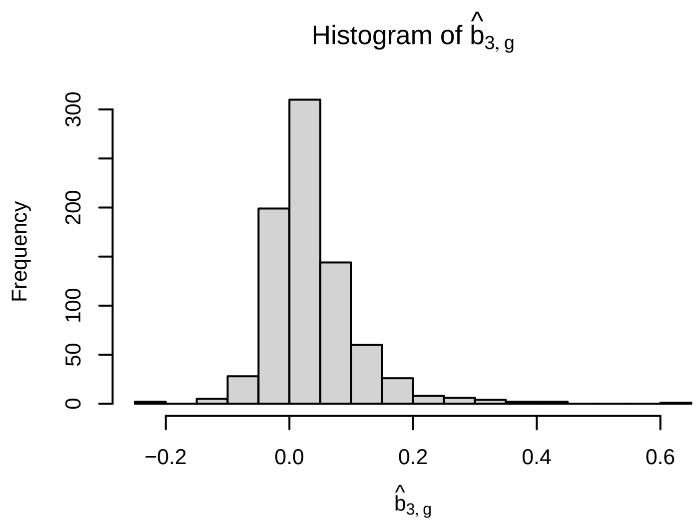
We add another bias term b_g to our model for genre effects:

$$Y_{u,i} \sim \mu + b_{1;i} + b_{2;u} + b_{3;g(i)} + \varepsilon_{u,i},$$

where $g(i)$ is the *combination* of genres for movie i. We approximate $b_{3;g}$ for each genre combination g as the mean of $\hat{b}_u = Y_{u;i} - \hat{\mu} - \hat{b}_{1;i} - \hat{b}_{2;u}$, averaged over all ratings in the training set where $g(i) = g$. The following code computes $\hat{b}_{3,g}$ for each g and plots these as a histogram:

```r
# Estimate genre effects
genre_biases <- edx_train |>
  left_join(movie_biases, by='movieId') |>
  left_join(user_biases, by='userId') |>
  group_by(genres) |>
  summarize(b_g = mean(rating - mu - b_i - b_u))
```

```r
# Plot a histogram of the genre effects
par(cex = 0.7)
hist(genre_biases$b_g, 30, xlab = TeX(r'[$\hat{b}_{3,g}$]'),
     main = TeX(r'[Histogram of $\hat{b}_{3,g}$]'))
```



The new model gives the following RMSE values when applied to the `edx_test` set:

```r
# Obtain predictions for the edx_test set
predicted_ratings <- edx_test |>
  left_join(movie_biases, by='movieId') |>
  left_join(user_biases, by='userId') |>
  left_join(genre_biases, by='genres') |>
  mutate(pred = mu + b_i + b_u + b_g) |> pull(pred)

# Compute RMSE and add to data.table
RMSEs <- RMSEs |>
  add_row(Method = "Movie + user + genre effects",
          RMSE = RMSE(predicted_ratings, edx_test$rating),
          "RMSE (clamped estimates)" = RMSE(clamp(predicted_ratings), edx_test$rating))
RMSEs[nrow(RMSEs),] |> kable(align='lrr', booktabs = T) |> row_spec(0, bold = T)
```

Method	**RMSE**	**RMSE (clamped estimates)**
Movie + user + genre effects	0.8643241	0.8641138

2.6 Adding a time effect

Consider a new model with the form

$$Y_{u,i} \sim \mu + b_{1;i} + b_{2;u} + b_{3;g(i)} + f(t_{u,i}) + \varepsilon_{u,i}.$$

where $t_{u,i}$ is a week index, such that the date of the oldest rating is defined as the start of Week 1.

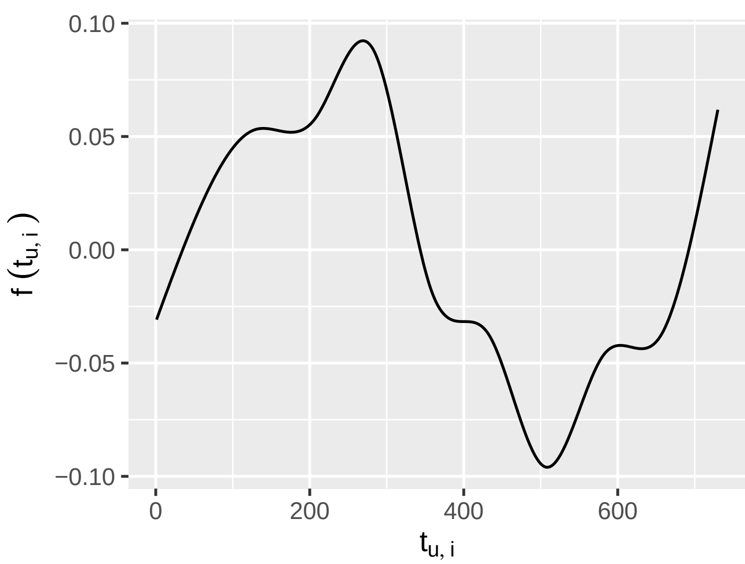
The new optimization problem minimizes

$$\mathrm{SE} = \sum_{(u,i)\in\mathcal{T}} \left[y_{u,i} - \mu - b_{1;i} - b_{2;u} - b_{3;g(i)} - f(t_{u,i})\right]^2.$$

Note the addition of the t subscript compared to the original problem formulation defined in Section 2.1, with $\mu_{t;u,i} = \mu - f(t_{u,i})$.

The following code defines $f(t)$ as the smoothed average rating on Week t, minus μ:

```r
# Add a week number to each rating in the edx_train and edx_test datasets
edx_train <- edx_train |>
  mutate(weekNum = (timestamp - min(timestamp)) |>
           as.numeric(unit = "days") |> {\(x) floor(x/7) + 1}() )
edx_test <- edx_test |>
  mutate(weekNum = (timestamp - min(timestamp)) |>
           as.numeric(unit = "days") |> {\(x) floor(x/7) + 1}() )

# Fit a smooth curve to the ratings as a function of time
fit <- mgcv::gam(rating ~ s(weekNum, bs = "cs"),
                 family = gaussian(), data = edx_train) # apply smoothing

# Evaluate the fitted curve for each week number
r <- seq(1,max(edx_train$weekNum))
f_t <- mgcv::predict.gam(fit, data.frame(weekNum = r)) - mu
rm(fit)

# Plot the fitted curve
ggplot(data.frame(weekNum = r, f_t), aes(weekNum, f_t)) + geom_line() +
  xlab(TeX(r'[$t_{u,i}$]')) + ylab(TeX(r'[$f\,(t_{u,i}\,)$]'))
```



We approximate $b_{t,g}$ for each genre combination g as the mean of $\hat{b}_u = Y_{u,i} - \hat{\mu} - \hat{b}_{1;i} - \hat{b}_{2;u} - b_{3;g(i)}$. Fitting the $b_{j;t}$'s $j = 1, 2, 3$, for the new model, we obtain RMSE values of:

```r
# Compute the biases

movie_biases_t <- edx_train |>
  mutate(f_t = f_t[weekNum]) |>
  group_by(movieId) |>
  summarize(b_i = mean(rating - mu - f_t))

user_biases_t <- edx_train |>
  mutate(f_t = f_t[weekNum]) |>
  left_join(movie_biases_t, by='movieId') |>
  group_by(userId) |>
  summarize(b_u = mean(rating - mu - b_i - f_t))

genre_biases_t <- edx_train |>
```

```r
  mutate(f_t = f_t[weekNum]) |>
  left_join(movie_biases_t, by='movieId') |>
  left_join(user_biases_t, by='userId') |>
  group_by(genres) |>
  summarize(b_g = mean(rating - mu - b_i - b_u - f_t))

# Obtain predictions for the edx_test set
predicted_ratings <- edx_test |>
  mutate(f_t = f_t[weekNum]) |>
  left_join(movie_biases_t, by='movieId') |>
  left_join(user_biases_t, by='userId') |>
  left_join(genre_biases_t, by='genres') |>
  mutate(pred = mu + b_i + b_u + b_g + f_t) |>
  pull(pred)

# Compute RMSE and add to data.table
RMSEs <- RMSEs |>
  add_row(Method = "Movie + user + genre + time effects",
          RMSE = RMSE(predicted_ratings, edx_test$rating),
          "RMSE (clamped estimates)" = RMSE(clamp(predicted_ratings), edx_test$rating))
RMSEs[nrow(RMSEs),] |> kable(align='lrr', booktabs = T) |> row_spec(0, bold = T)
```

Method	**RMSE**	**RMSE (clamped estimates)**
Movie + user + genre + time effects	0.8641266	0.8639174

2.7 Adding L_2 regularization

To improve our model further, we can add L_2 regularization. Whereas the previous model fitting procedure minimizes

$$\mathrm{SE} = \sum_{(u,i)\in\mathcal{T}} \left[y_{u,i} - \mu - b_{1;i} - b_{2;u} - b_{3;g(i)} - f(t_{u,i})\right]^2,$$

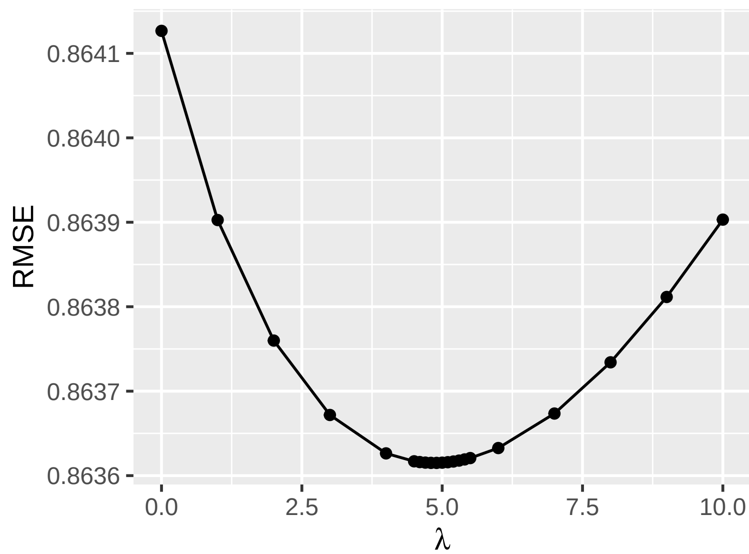
in this section we add a penalty term such that the new expression to minimize is as follows:

$$\mathrm{SE} + \lambda \sum_j \|b_j\|_2^2.$$

Fitting the regularized model to the training set for different λ, and using the test set for RMSE calculation, we obtain the following plot of RMSE against λ.

```r
# List of regularization parameter values to try.
# Since I know the approximate optimal value, I added more points
# in this range.
lambdas <- c(0,1,2,3,4,seq(4.5,5.5,0.1),6,7,8,9,10)

# Compute RMSE values for each lambda using the *test set.
rmses <- sapply(lambdas, function(l){
  message("lambda = ", l)

  # Compute movie, user, genre, and time effects using the test set.
  # Note that f_t here refers to the variable f_t and not the f_t column in
  # any of the data.tables.
  movie_biases_reg <-
    edx_train[, .(b_i = sum(rating - mu - f_t[weekNum])/(.N+l)), by = 'movieId']

  temp <- movie_biases_reg[edx_train, on = 'movieId']
  user_biases_reg <-
    temp[, .(b_u = sum(rating - mu - b_i - f_t[weekNum])/(.N+l)), by = 'userId']
```

```
  temp <- user_biases_reg[temp, on = 'userId']
  genre_biases_reg <-
    temp[, .(b_g = sum(rating - mu - b_i - b_u - f_t[weekNum])/(.N+l)), by = 'genres']

  # Generate predictions
  predicted_ratings <- genre_biases_reg[
    user_biases_reg[
      movie_biases_reg[
        edx_test, on = 'movieId'],
      on = 'userId'],
    on = 'genres'] |>
    mutate(pred = mu + b_i + b_u + b_g + f_t[weekNum]) |>
    pull(pred)

  # Compute RMSE
  return(RMSE(predicted_ratings, edx_test$rating))
})

# Plot RMSE against lambda
par(cex = 0.7)
qplot(lambdas, rmses, xlab = TeX(r'($\lambda)'), ylab = 'RMSE', geom = c('point', 'line'))
```



The optimal value of λ is thus:

```
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.9
```

Fitting the regularized model one last time and computing the RMSE on **`edx_test`**, we obtain:

```
movie_biases_reg <-
  edx_train[, .(b_i = sum(rating - mu - f_t[weekNum])/(.N+lambda)), by = 'movieId']

temp <- movie_biases_reg[edx_train, on = 'movieId']
user_biases_reg <-
  temp[, .(b_u = sum(rating - mu - b_i - f_t[weekNum])/(.N+lambda)), by = 'userId']

temp <- user_biases_reg[temp, on = 'userId']
```

```
genre_biases_reg <-
  temp[, .(b_g = sum(rating - mu - b_i - b_u - f_t[weekNum])/(.N+lambda)), by = 'genres']

# Generate predictions for the *edx_test* set.
predicted_ratings_reg <- genre_biases_reg[
  user_biases_reg[
    movie_biases_reg[
      edx_test, on = 'movieId'],
    on = 'userId'],
  on = 'genres'] |>
  mutate(pred = mu + b_i + b_u + b_g + f_t[weekNum]) |>
  pull(pred)

rm(temp)

# Compute RMSE and add to data.table
RMSEs <- RMSEs |>
  add_row(Method = "Movie + user + genre + time effects (regularized)",
          RMSE = RMSE(predicted_ratings_reg, edx_test$rating),
          "RMSE (clamped estimates)" =
            RMSE(clamp(predicted_ratings_reg), edx_test$rating))
RMSEs[nrow(RMSEs),] |> kable(align='lrr', booktabs = T) |> row_spec(0, bold = T)
```

Method	**RMSE**	**RMSE (clamped estimates)**
Movie + user + genre + time effects (regularized)	0.8636151	0.8634932

2.8 Section summary

The table of RMSEs for all models considered in this section is below.

```
RMSEs |> kable(align='lrr', booktabs = T, linesep = "") |> row_spec(0, bold = T)
```

Method	**RMSE**	**RMSE (clamped estimates)**
Mean only	1.0600537	1.0600537
Movie effects	0.9429615	0.9429615
Movie + user effects	0.8646843	0.8644818
Movie + user + genre effects	0.8643241	0.8641138
Movie + user + genre + time effects	0.8641266	0.8639174
Movie + user + genre + time effects (regularized)	0.8636151	0.8634932

The results demonstrate that each added feature has reduced the RMSE, as well as adding regularization and clamping; however, there are diminishing returns as each effect is added to the model.

3 Funk's matrix factorization algorithm

In this section, we consider Funk's matrix factorization (MF) algorithm (Funk 2006; Koren, Bell, and Volinsky 2009) for rating prediction. We use the model $Y \sim P + \sim UV^{\mathrm{T}} + \varepsilon$ where:

- Y is the $N_{\mathrm{U}} \times N_{\mathrm{M}}$ rating matrix, i.e., with N_u users and N_i movies,
- P represents the predictions from best model of the previous section,
- U and V are $N_u \times k$ and $N_i \times k$ matrices, respectively, where k is the number of *latent features* to be found.

Unknown ratings $Y_{u,i}$ can thus be estimated as $P_{u,i} + U_u V_i^{\mathrm{T}}$. The parameter k is also the *rank* of matrix UV; i.e. UV is a rank-k approximation of the residual matrix $Y - P$.

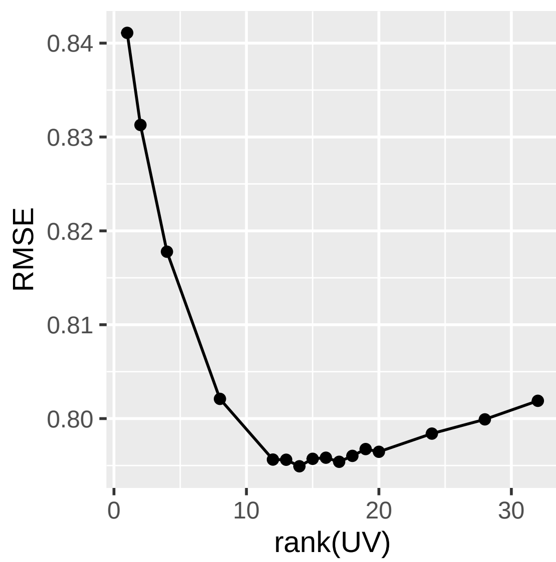
Funk's MF estimates U and V using gradient descent, but operating only on the known ratings. First, U and V are seeded with random values. Then, for each epoch, the algorithm iterates over all known ratings (i, j) in the training set and updates the feature matrices as follows:

$$e_{ij} = Y_{ij} - P_{ij} - U_i V_j^{\mathrm{T}}$$
$$U_i \leftarrow U_i + \gamma(e_{ij} V_j - \lambda U_i)$$
$$V_j \leftarrow V_i + \gamma(e_{ij} U_i - \lambda V_j)$$

where γ is the learning rate and λ is a regularization parameter. In this report, these are set to 0.02 and 0.001, respectively, in accordance to guidance from Funk (2006). The code for the Funk MF implementation used in this report can be found in Appendix A.

```r
# Residuals from previous best model
previous_train <- genre_biases_reg[
  user_biases_reg[
    movie_biases_reg[
      edx_train, on = 'movieId'],
    on = 'userId'],
  on = 'genres'] |>
  mutate(pred = mu + b_i + b_u + b_g + f_t[weekNum]) |>
  pull(pred)

residuals_train <- edx_train$rating - previous_train

# Test set predictions for previous best model
previous_test <- genre_biases_reg[
  user_biases_reg[
    movie_biases_reg[
      edx_test, on = 'movieId'],
    on = 'userId'],
  on = 'genres'] |>
  mutate(pred = mu + b_i + b_u + b_g + f_t[weekNum]) |>
  pull(pred)

# Obtain new movie and user indices **without gaps**, and save the mappings
Uidx <- numeric(max(edx_train$userId))
Uidx[unique(edx_train$userId)] = seq(uniqueN(edx_train$userId))

Vidx <- numeric(max(edx_train$movieId))
Vidx[unique(edx_train$movieId)] = seq(uniqueN(edx_train$movieId))

# Funk matrix factorization. See C++ source for full documentation.
# Values for regCoef and learningRate are as suggested by [Funk 2006].
Rcpp::sourceCpp("svd.cpp")
funk <- function(Uidx, Vidx, residuals, nFeatures, steps = 500,
                 regCoef = 0.02, learningRate = 1e-3) {

  # Change Uidx and Vidx to 0-based, for C++ only.
  funkCpp(Uidx[edx_train$userId] - 1,
          Vidx[edx_train$movieId] - 1,
          residuals_train,
          nFeatures, steps, regCoef, learningRate)
}

# Compute RMSE values for varying number of MF features.
set.seed(1)
if (!file.exists('funk_tuning.Rdata')) {
  nFeatures <- c(1, 2, 4, 8, seq(12,20), 24, 28, 32)
  rmses <- sapply(nFeatures, \(nF){

    message(nF, ' features')
```

```
    # Run Funk MF
    set.seed(1)
    funkResult <- funk(Uidx, Vidx, residuals_train, nFeatures = nF, steps = 500)
    U <- funkResult$U
    V <- funkResult$V

    # Uidx[u] is the row index of user u in matrix U
    # Vidx[v] is the row index of movie v in matrix V
    predicted_ratings_funk <- edx_test |>
      mutate(pred = previous_test +
               map2_dbl(userId, movieId, \(u,v) U[Uidx[u],] %*% V[Vidx[v],])) |>
      pull(pred)
    rmse <- RMSE(predicted_ratings_funk, edx_test$rating)

    message(rmse,'\n')
    return(rmse)
  })

  save(nFeatures,rmses, file = 'funk_tuning.Rdata')
}
set.seed(1)

load('funk_tuning.Rdata')
par(cex = 0.7)
qplot(nFeatures, rmses, xlab = 'rank(UV)', ylab = 'RMSE', geom = c('point','line'))
```




```
nFeaturesOpt <- nFeatures[which.min(rmses)]
nFeaturesOpt
```

```
## [1] 14
```

Using the new model with $k = 14$ to predict ratings for the `edx_test` gives the following RMSE values:

```
# Run Funk MF
set.seed(1)
funkResult <- funk(Uidx, Vidx, residuals_train, nFeatures = nFeaturesOpt, steps = 500)
U <- funkResult$U
V <- funkResult$V
save(nFeaturesOpt, funkResult, file = 'funk.Rdata')
```

```r
set.seed(1)

load('funk.Rdata')

# Uidx[u] is the row index of user u in matrix U
# Vidx[v] is the row index of movie v in matrix V
predicted_ratings_funk <- edx_test |>
  mutate(pred = previous_test +
           map2_dbl(userId, movieId, \(u,v) U[Uidx[u],] %*% V[Vidx[v],])) |>
  pull(pred)
rmse <- RMSE(predicted_ratings_funk, edx_test$rating)

# Compute RMSE and add to data.table
RMSEs <- RMSEs |>
  add_row(Method = "Section 2 best model + Matrix factorization",
          RMSE = RMSE(predicted_ratings_funk, edx_test$rating),
          "RMSE (clamped estimates)" = RMSE(clamp(predicted_ratings_funk), edx_test$rating))

RMSEs[nrow(RMSEs),] |> kable(align='lrr', booktabs = T) |> row_spec(0, bold = T)
```

Method	**RMSE**	**RMSE (clamped estimates)**
Section 2 best model + Matrix factorization	0.7949206	0.7939817

The RMSEs of all models in this report, evaluated using `edx_test`, are as follows:

```r
RMSEs |> kable(align='lrr', booktabs = T, linesep = "") |> row_spec(0, bold = T)
```

Method	**RMSE**	**RMSE (clamped estimates)**
Mean only	1.0600537	1.0600537
Movie effects	0.9429615	0.9429615
Movie + user effects	0.8646843	0.8644818
Movie + user + genre effects	0.8643241	0.8641138
Movie + user + genre + time effects	0.8641266	0.8639174
Movie + user + genre + time effects (regularized)	0.8636151	0.8634932
Section 2 best model + Matrix factorization	0.7949206	0.7939817

We now “submit” our best model, i.e.

$$Y_{u,i} \sim \mu + b_{1;i} + b_{2;u} + b_{3;g(i)} + f(t_{u,i}) + UV^{\mathrm{T}} + \varepsilon_{u,i}, \tag{1}$$

with parameters `mu`, `movie_biases_reg`, `user_biases_reg`, `genre_biases_reg`, `f_t`, `U`, and `V`, for final validation.

4 Final validation

We select our best model (1) for validation against the `validation` dataset.

```r
save(mu, movie_biases_reg, user_biases_reg, genre_biases_reg,
     f_t, Uidx, Vidx, U, V, file = 'FINAL_model.Rdata')
```

The number of parameters in the model is:

```r
nrow(movie_biases_reg) + nrow(user_biases_reg) + nrow(genre_biases_reg) +
  length(f_t) + length(U) + length(V)
```

```
## [1] 1209852
```

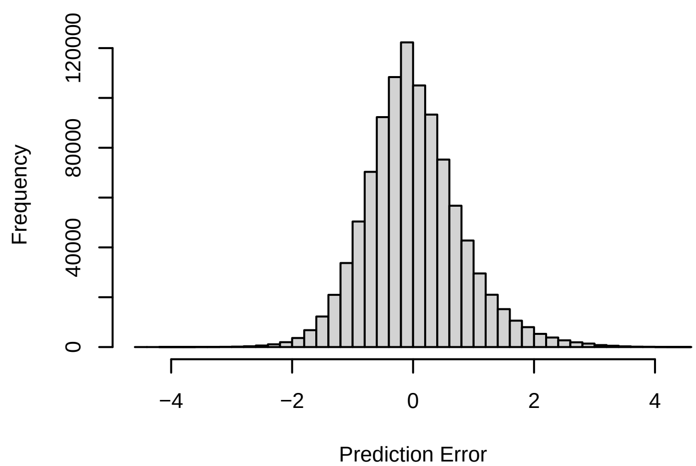
The final RMSE computed with the `validation` set is:

```r
predicted_ratings_FINAL_VALIDATION <- validation |>
  mutate(weekNum = (timestamp - min(timestamp)) |>
           as.numeric(unit = "days") |> {\(x) floor(x/7) + 1}() ) |>
  mutate(f_t = f_t[weekNum]) |>
  left_join(movie_biases_reg, by='movieId') |>
  left_join(user_biases_reg, by='userId') |>
  left_join(genre_biases_reg, by='genres') |>
  mutate(pred = mu + b_i + b_u + b_g + f_t +
           map2_dbl(userId, movieId, \(u,v) U[Uidx[u],] %*% V[Vidx[v],])) |>
  pull(pred) |> clamp()

# Compute RMSE and add to data.table
RMSE(predicted_ratings_FINAL_VALIDATION, validation$rating)
```

```
## [1] 0.7941947
```

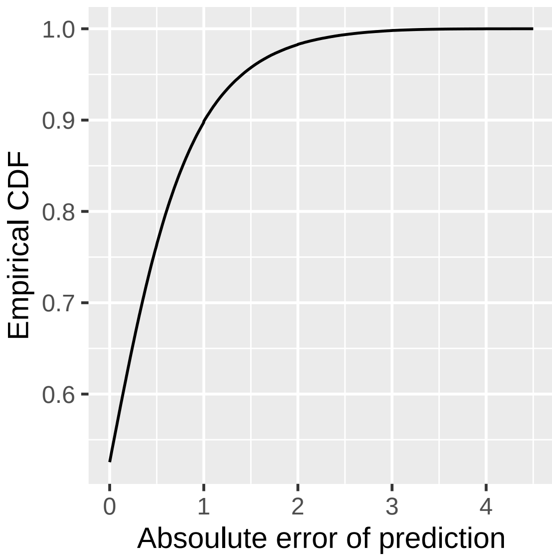
The plot below plots a histogram of prediction errors:

```r
par(cex = 0.7)
hist(predicted_ratings_FINAL_VALIDATION - validation$rating, 50,
     xlab = 'Prediction Error', main = '')
```



Below is a plot of the cumulative distribution of the absolute error:

```r
the_ecdf <- ecdf(predicted_ratings_FINAL_VALIDATION - validation$rating)
par(cex = 0.7)
qplot(seq(0,4.5,0.001), the_ecdf(seq(0,4.5,0.001)),
     xlab = 'Absoulute error of prediction', ylab = 'Empirical CDF', geom = 'line')
```



The proportion of predictions are within half a star of the actual rating is:

```
mean(abs(predicted_ratings_FINAL_VALIDATION - validation$rating) < 0.5)
```

```
## [1] 0.5169635
```

5 Concluding remarks

In this project, we train a recommender system to predict movie ratings on a scale from 0.5 to 5, using the Movielens 10M (GroupLens 2009) dataset. Our final model considers user, movie, genre, and time-based biases, and uses Funk's matrix factorization to approximate the residuals after these effects have been removed from the ratings. The RMSE achieved by our final model, as evaluated using the validation partition, is 0.7939817.

Note that the effect of adding genre and time-based biases was small. In particular, the genres of a movie can be uniquely determined from its movieID, and most movies have been rated many times, decreasing the importance of genre information. For the same reason, adding the year of release of each movie as a model feature is also unlikely to significantly improve the results. However, genre and time-based information will prove useful for predicting ratings of *new* movies, where a movie bias cannot be computed (using a zero value is the likely best solution). In this case, adding the year of release as an additional model feature likely *would* improve prediction accuracy. Another possible feature we could have used is the age of a movie *at the time it was rated*.

A consideration is the fact that while this project attempts to minimize the error of the raw ratings, a possibly better approach may be binary: would a user like a movie they have not yet watched, if that movie were recommended to them? If we assume a user enjoys a movie if they rate it 3.5 stars or higher, then the confusion matrix as computed on the validation set is:

```
confusionMatrix(as.factor(ifelse(predicted_ratings_FINAL_VALIDATION ≥ 3.5, 'Good', 'Bad')),
                as.factor(ifelse(validation$rating ≥ 3.5, 'Good', 'Bad')),
                positive = 'Good')
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction     Bad   Good
##       Bad  301123 140407
##       Good 110332 448137
##
##                Accuracy : 0.7493
##                  95% CI : (0.7484, 0.7501)
```

```
##     No Information Rate : 0.5885
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4879
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7614
##             Specificity : 0.7318
##          Pos Pred Value : 0.8024
##          Neg Pred Value : 0.6820
##              Prevalence : 0.5885
##          Detection Rate : 0.4481
##    Detection Prevalence : 0.5585
##       Balanced Accuracy : 0.7466
##
##        'Positive' Class : Good
##
```

The accuracy of our model is about three-quarters, with approximately equal sensitivity and specificity.

Furthermore, note that while the ratings in the Movielens dataset are discrete, the generated predictions are not. If only discrete predictions are allowed, then a series of thresholds may be fitted to our current model for binning (these thresholds do not have to be a half-star apart and can instead be based on the distribution of true and predicted ratings). It remains to be seen how such an approach would affect the accuracy of our model, as while correct binning decreases the error of a prediction, incorrect binning may instead increase the error of a rating. For example, for a prediction of 3.6 that is binned to 3.5, the error decreases from 0.1 to 0 given a true rating of 0.5, but increases from 0.4 to 0.5 given a true rating of 4.0.

```
# NOTRUN: generate R script from project
knitr::purl('_merged.Rmd')
```

References

Funk, Simon. 2006. "Netflix Update: Try This at Home." http://sifter.org/~simon/journal/20061211.html.

GroupLens. 2009. "MovieLens 10M Dataset." https://grouplens.org/datasets/movielens/10m/.

Koren, Yehuda, Robert Bell, and Chris Volinsky. 2009. "Matrix Factorization Techniques for Recommender Systems." *Computer* 42 (8): 30–37. https://doi.org/10.1109/mc.2009.263.

A Code listing: `svd.cpp`

```
// [[Rcpp::depends(RcppArmadillo)]]
// [[Rcpp::depends(RcppProgress)]]

#include <RcppArmadillo.h>
#include <progress.hpp>
#include <progress_bar.hpp>

/**
 * @brief Simon Funk's Matrix Factorization.
 *
 * Approximate Y as U*V^T where U and V each have @p nFeatures columns.
 *
 * @param coo_i User indexes of the rating matrix Y.
 * @param coo_j Movie indexes of the rating matrix Y.
 * @param coo_x Ratings in the rating matrix Y.  Note Y is a sparse matrix, where
 *     a zero represents no rating given.
```

```
 * @param nFeatures the number of features to use, i.e. the number of columns in U and V.
 * @steps Number of epochs. Each epoch refines the U and V estimates by iterating
 *     through all known ratings once.
 * @regCoef Regularization coefficient, prevents overfitting.
 * @learningRate learning rate of gradient descent.
 *
 * @return An @c RCpp::list object containing U and V.
 *
 * @see https://sifter.org/~simon/journal/20061211.html
 * @see https://github.com/ludovikcoba/rrecsys/
 */
// [[Rcpp::export]]
Rcpp::List funkCpp(
    Rcpp::NumericVector coo_i,
    Rcpp::NumericVector coo_j,
    Rcpp::NumericVector coo_x,
    int nFeatures,
    int steps,
    double regCoef,
    double learningRate
)
{
  int nUsers = Rcpp::max(coo_i)+1; // number of users
  int nItems = Rcpp::max(coo_j)+1; // number of movies (items)
  int nRatings = coo_x.size();     // number of known ratings

  // Seed U and V with random values
  arma::mat U(nUsers, nFeatures, arma::fill::randu);
  arma::mat V(nItems, nFeatures, arma::fill::randu);
  U *= sqrt(0.5/nFeatures);
  V *= sqrt(0.5/nFeatures);

  // Diagnostics logging
  Rcpp::Rcerr << "nUsers:" << nUsers << ", ";
  Rcpp::Rcerr << "nItems:" << nItems << ", ";
  Rcpp::Rcerr << "nRatings:" << nRatings << std::endl;

  // Progress bar for R console
  Progress p(steps, true);

  // Main loop
  for (int ss = 0; ss < steps; ss++) {

    // Kill program if user has requested it (Ctrl+C in most consoles)
    Rcpp::checkUserInterrupt();

    // iterate over known ratings
    for (int r = 0; r < nRatings; r++) {
      int i = coo_i[r]; // user index
      int j = coo_j[r]; // item index
      double err = coo_x[r] - arma::dot(U.row(i), V.row(j)); // prediction error

      // update features
      U.row(i) += learningRate * (err*V.row(j) - regCoef*U.row(i));
      V.row(j) += learningRate * (err*U.row(i) - regCoef*V.row(j));
    }

    // Report progress
```

```
    p.increment();
  }
  Rcpp::Rcerr << std::endl; // add gap between progress bars of multiple runs

  // Return list(U,V)
  Rcpp::List ret;
  ret["U"] = U;
  ret["V"] = V;
  return ret;
}
```

B Session info

```
sessionInfo()
```

```
## R version 4.1.3 (2022-03-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 22000)
##
## Matrix products: default
##
## Random number generation:
##  RNG:     Mersenne-Twister
##  Normal:  Inversion
##  Sample:  Rounding
##
## locale:
## [1] LC_COLLATE=English_Hong Kong SAR.1252
## [2] LC_CTYPE=English_Hong Kong SAR.1252
## [3] LC_MONETARY=English_Hong Kong SAR.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_Hong Kong SAR.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] lubridate_1.8.0   caret_6.0-91      lattice_0.20-45   forcats_0.5.1
##  [5] stringr_1.4.0     dplyr_1.0.8       purrr_0.3.4       readr_2.1.2
##  [9] tidyr_1.2.0       tibble_3.1.6      ggplot2_3.3.5     tidyverse_1.3.1
## [13] patchwork_1.1.1   latex2exp_0.9.4   data.table_1.14.2 kableExtra_1.3.4
## [17] knitr_1.38        magrittr_2.0.3    pacman_0.5.1
##
## loaded via a namespace (and not attached):
##  [1] colorspace_2.0-3        ellipsis_0.3.2          class_7.3-20
##  [4] RcppArmadillo_0.11.0.0.0 fs_1.5.2                proxy_0.4-26
##  [7] rstudioapi_0.13         listenv_0.8.0           farver_2.1.0
## [10] bit64_4.0.5             prodlim_2019.11.13      fansi_1.0.3
## [13] xml2_1.3.3              codetools_0.2-18        splines_4.1.3
## [16] jsonlite_1.8.0          pROC_1.18.0             broom_0.7.12
## [19] dbplyr_2.1.1            compiler_4.1.3          httr_1.4.2
## [22] backports_1.4.1         assertthat_0.2.1        Matrix_1.4-1
## [25] fastmap_1.1.0           cli_3.2.0               htmltools_0.5.2
## [28] tools_4.1.3             gtable_0.3.0            glue_1.6.2
## [31] reshape2_1.4.4          Rcpp_1.0.8.3            cellranger_1.1.0
## [34] raster_3.5-15           vctrs_0.4.0             svglite_2.1.0
## [37] nlme_3.1-157            iterators_1.0.14        timeDate_3043.102
```

```
## [40] gower_1.0.0            xfun_0.30              RcppProgress_0.4.2
## [43] globals_0.14.0         rvest_1.0.2            lifecycle_1.0.1
## [46] future_1.24.0          terra_1.5-21           MASS_7.3-56
## [49] scales_1.1.1           ipred_0.9-12           vroom_1.5.7
## [52] hms_1.1.1              parallel_4.1.3         yaml_2.3.5
## [55] rpart_4.1.16           stringi_1.7.6          highr_0.9
## [58] foreach_1.5.2          e1071_1.7-9            hardhat_0.2.0
## [61] lava_1.6.10            rlang_1.0.2            pkgconfig_2.0.3
## [64] systemfonts_1.0.4      evaluate_0.15          labeling_0.4.2
## [67] recipes_0.2.0          bit_4.0.4              tidyselect_1.1.2
## [70] parallelly_1.30.0      plyr_1.8.7             bookdown_0.25
## [73] R6_2.5.1               generics_0.1.2         DBI_1.1.2
## [76] pillar_1.7.0           haven_2.4.3            withr_2.5.0
## [79] mgcv_1.8-40            survival_3.3-1         sp_1.4-6
## [82] nnet_7.3-17            future.apply_1.8.1     modelr_0.1.8
## [85] crayon_1.5.1           utf8_1.2.2             tzdb_0.3.0
## [88] rmarkdown_2.13         grid_4.1.3             readxl_1.4.0
## [91] ModelMetrics_1.2.2.2   reprex_2.0.1           digest_0.6.29
## [94] webshot_0.5.2          stats4_4.1.3           munsell_0.5.0
## [97] viridisLite_0.4.0
```

```
tidyverse::tidyverse_conflicts()
```

```
## -- Conflicts ────────────────────────────────────────── tidyverse_conflicts() --
## x lubridate::as.difftime() masks base::as.difftime()
## x dplyr::between()         masks data.table::between()
## x lubridate::date()        masks base::date()
## x tidyr::extract()         masks magrittr::extract()
## x dplyr::filter()          masks stats::filter()
## x dplyr::first()           masks data.table::first()
## x dplyr::group_rows()      masks kableExtra::group_rows()
## x lubridate::hour()        masks data.table::hour()
## x lubridate::intersect()   masks base::intersect()
## x lubridate::isoweek()     masks data.table::isoweek()
## x dplyr::lag()             masks stats::lag()
## x dplyr::last()            masks data.table::last()
## x caret::lift()            masks purrr::lift()
## x lubridate::mday()        masks data.table::mday()
## x lubridate::minute()      masks data.table::minute()
## x lubridate::month()       masks data.table::month()
## x lubridate::quarter()     masks data.table::quarter()
## x lubridate::second()      masks data.table::second()
## x purrr::set_names()       masks magrittr::set_names()
## x lubridate::setdiff()     masks base::setdiff()
## x purrr::transpose()       masks data.table::transpose()
## x lubridate::union()       masks base::union()
## x lubridate::wday()        masks data.table::wday()
## x lubridate::week()        masks data.table::week()
## x lubridate::yday()        masks data.table::yday()
## x lubridate::year()        masks data.table::year()
```