MovieLens Project Report

Thomas D. Pellegrin

December 2023

Abstract

This report presents the methods and results from a capstone assignment of the [https://www.edx.org/certificates/professional-certificate/harvardx-data-science](HarvardX PH125.9 "Data Science" course) available on the online learning platform edX. The assignment used an archival dataset of user ratings of movies named *Movielens*. The data were first split into training and final holdout sets. Exploratory analysis conducted on the training set revealed strong movie and user effects. Next, a recommendation model from the *recosystem* package of the R statistical computing environment was trained against the training set. The trained model was then used to predict ratings in the final holdout set of users and movies. Lastly, the predicted ratings were compared to the final holdout set's actual ratings. The resulting root mean square error (RMSE) is 0.7778, which is lower than the target RMSE of 0.8649.

Introduction

The final module of the [https://www.edx.org/certificates/professional-certificate/harvardx-data-science](HarvardX PH125.9 "Data Science" course) from edX requires certification candidates to independently develop and submit a capstone project comprised of a quiz and two assignments.

The first assignment's goal is to create a movie recommendation system against the MovieLens dataset, which is comprised of ten million rows observations of movie ratings by users of the online recommendation service https://movielens.org/ collected between January 1995 and January 2009.

The present report describes the steps followed to conduct this assignment. The **Methods** section describes how the data were downloaded, transformed, and explored. Several hypotheses were formed about the relationship between predictor variables (such as movie and user) and the response variable (the rating). Biases related to each movie, user, genre, year of release, year of rating, and number of ratings per movie were measured as a deviation from the mean rating for each grouping of predictor variable. The respective contributions of those biases to improving a naive prediction of movie ratings from the final test dataset was measured using the root mean square error (RMSE), which is the mean difference between predicted and actual values. The movie and user effects were found to be the strongest determinants of RMSE minimization, along with a the number of ratings per movie.

Following this exploratory analysis, a recommendation model from R's *recosystem* package was tuned and trained on a sparse matrix in triplet form of userId, movieId, and ratings. The trained model was then used to predict ratings for previously-unseen user and movie combinations from the final holdout test set. The model was found to far exceed the performance of the manual predictions accounting for the biases in the predictor variables.

Methods

Data preparation

The *edx* dataset was generated using R code **adapted** from the course instructions. First, the dataset was downloaded from the [https://grouplens.org/datasets/movielens/10m/][GroupLens] website as a 63 MB compressed archive

```
# Define the data source
dl  <- "ml-10M100K.zip"
url <- "https://files.grouplens.org/datasets/movielens/ml-10m.zip"

# Download the data only if not already downloaded
if(!file.exists(dl)) { download.file(url, dl) }
```

Next, the *movies.dat* and *ratings.dat* files were extracted from the compressed archive and their respective data frames joined into one *movielens* data frame using *movieId* as the key. Four movies were found to have no rating and were removed from the dataset.

```
# Extract and join the movies and ratings
movielens <- left_join(

  read.table(
    text = gsub(
      x           = readLines(con = unzip(dl, "ml-10M100K/movies.dat")),
      pattern     = "::",
      replacement = ";",
      fixed       = TRUE
    ),
    sep           = ";",
    col.names     = c("movieId", "title", "genres"),
    colClasses    = c("integer", "character", "character"),
    quote         = "",
    comment.char  = "" # one movie title has a pound sign so we need this
  ),

  read.table(
    text = gsub(
      x           = readLines(con = unzip(dl, "ml-10M100K/ratings.dat")),
      pattern     = "::",
      replacement = ";",
      fixed       = TRUE
    ),
    sep        = ";",
    col.names  = c("userId", "movieId", "rating", "timestamp"),
    colClasses = c("integer", "integer", "numeric", "integer"),
    quote         = ""
  ),

  by = "movieId"

) |> na.omit()
```

Then, the data were partitioned into a training set (*edx_train*) of approximately nine million rows and a test set (*final_holdout_test*) of the remaining one million rows. The partition was done using the *caret* package's *createDataPartition* function. For reproducibility of the partition and results, R's random number generator was given a fixed seed value of 1.

Two semi-joins were performed in assembling the final test set to ensure that all *userId* and *movieId* values present in the test set were also present in the training set.

The purpose of the training set is to describe and explore the data and to tune and train the recommendation model. The purpose of the test set is to evaluate the recommendation system's performance against the course's grading rubric at the very end of the assignment. At no point was the test set used to tune or train the recommendation model; it was treated as "hidden data" available only after completion of the model.

```
# Set a seed for reproducibility
suppressWarnings(set.seed(1, sample.kind = "Rounding"))

# Partition the data. Test set will be 10% of MovieLens data
test_index <- createDataPartition(
  y     = movielens$rating,
  times = 1,
  p     = 0.1,
  list  = FALSE
)

# Separate the training and temporary test datasets
edx_train <- movielens[-test_index, ]
edx_temp  <- movielens[test_index, ]

# The temporary test set may now contain rows with a userId and/or movieId that
# are no longer present in the training set. This would cause problems when
# comparing rating predictions based on a userId and/or movieId bias. So, we
# need to temporarily remove any row in the final test set that doesn't have a
# matching userId and/or movieId in the training set.
final_holdout_test <- edx_temp |>
  semi_join(edx_train, by = "movieId") |>
  semi_join(edx_train, by = "userId")

# We add the row(s) removed from the final test set back into the training set
edx_train <- rbind(
  edx_train,                              # The training set partitioned earlier
  anti_join(edx_temp, final_holdout_test) # The row(s) removed from the final test set
)
```

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

Lastly, objects no longer needed for the rest of the analysis were removed to free up memory.

```
# Remove unnecessary objects from memory
rm(dl, test_index, edx_temp, movielens)
```

Exploratory analysis

Next, the *edx_train* data were described. The dataset consists of individual ratings assigned by a user to a movie. It contains 9,000,056 observations of 6 variables, which are described in turn hereafter.

movieId

An integer representation of the unique identifier of a movie, ranging from 1 to 65,133. Based on those IDs, there are 10,677 unique movies in the training set. Each movie was rated an average of 843 times.

title

A character representation of the title of the movie, with the year of release contained between brackets at the end. The year of release might be a predictor variable of the response variable *rating*. Therefore, it was extracted, assigned to a new integer variable *year_movie*, and removed from the title using the following code:

```
# Release year might be a predictor, so extract it into a new integer variable
edx_train <- edx_train |>
  mutate(
    year_movie  = as.integer(str_sub(title, start = -5L, end = -2L)),
    title       = str_sub(title, end = -8L)
  )
```

genres

A character representation of all the genres that a movie belongs to, concatenated and delimited by a pipe ("|") character. There are 20 unique genres in the training set, including one undefined genre. In their concatenated form, however, there are 797 combinations of genres in the *genres* variable. Either form might be a predictor of the response variable *rating*.

Figure 5 shows the distribution of genres in the training set. The drama genre is by far the most represented, followed by comedy, thriller, and romance. The IMAX, film-noir, and Western are the least represented genres.

```
edx_train |>
  distinct(title, .keep_all = TRUE) |>
  separate_rows(genres, sep = "\\|") |>
  count(genres, sort = TRUE) |>
  mutate(genres = factor(genres, levels = genres[order(n)])) |>
  ggplot(aes(x = genres, y = n)) +
  geom_col() +
  geom_text(
    aes(
      label = paste(
        format(x = n, big.mark = ","),
        " (",
        label_percent(accuracy = 0.1)(n / sum(n)),
        ")",
        sep = ""
      )
    ),
    nudge_y = 300L
  ) +
  coord_flip() +
  labs(x = "Genres", y = "Count of movies")
```

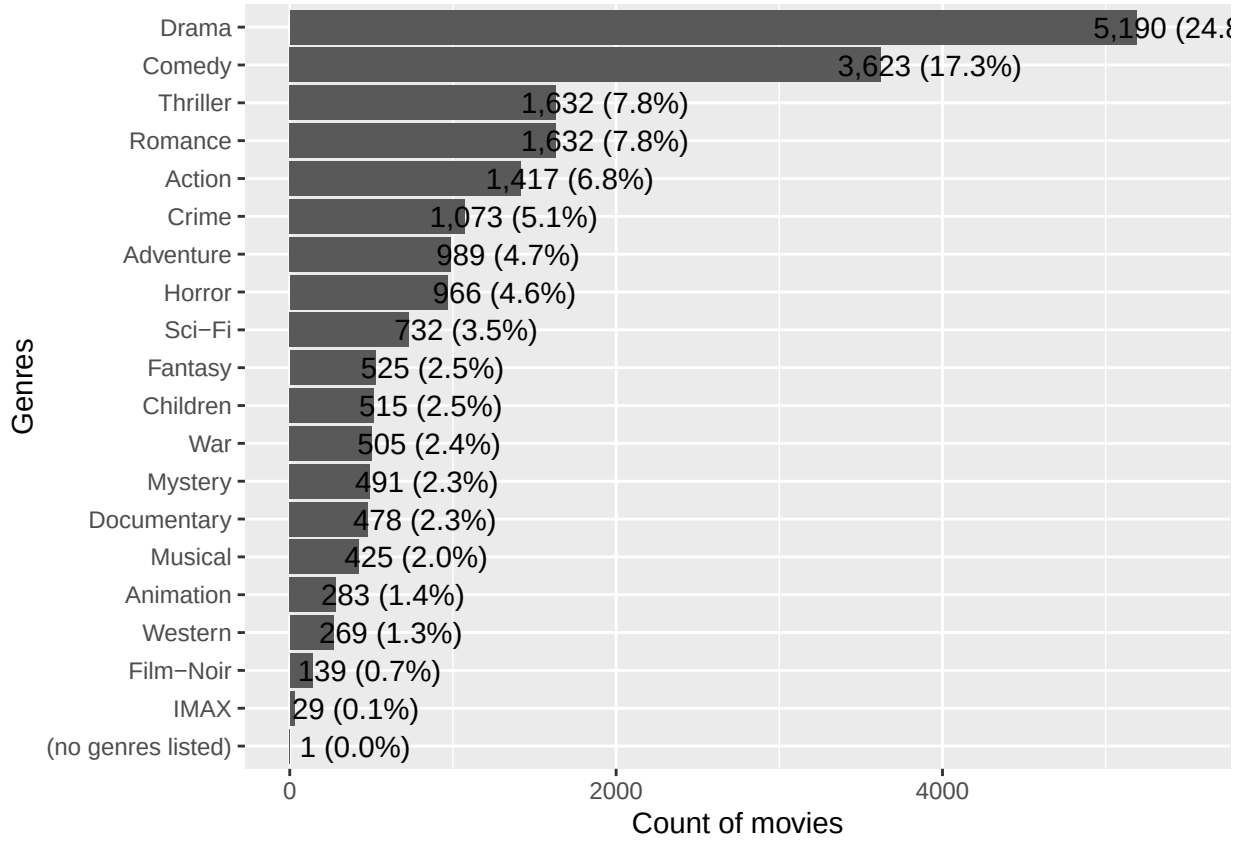

Figure 1: Histogram of genres

userId

An integer representation of the unique identifier of a user, ranging from 1 to 71,567. Based on those IDs, there are 69,878 unique users in the training set. Each user rated an average of 129 movies.

rating

A numerical representation of the movie's rating assigned by the user, on a Likert scale in 0.5 increments ranging from 0.5 to 5. This is the response variable that we must predict given new data (the final holdout test set).

Figure 5 shows the distribution of ratings in the training set. The most frequent rating (mode) is 4 (28.8%). The least frequent rating is 0.5 (0.9%). The mean of all the ratings is 3.51. There is no zero rating.

```
# Histogram of ratings
edx_train |>
  group_by(rating) |>
  count(rating, sort = FALSE) |>
  ggplot(aes(x = rating, y = n)) +
  geom_col() +
  geom_text(
    aes(label = label_percent(accuracy = 0.1)(n / sum(n))),
    nudge_y = 10^6 / 7
  ) +
  coord_flip() +
  labs(x = "Ratings", y = "Count of ratings")
```

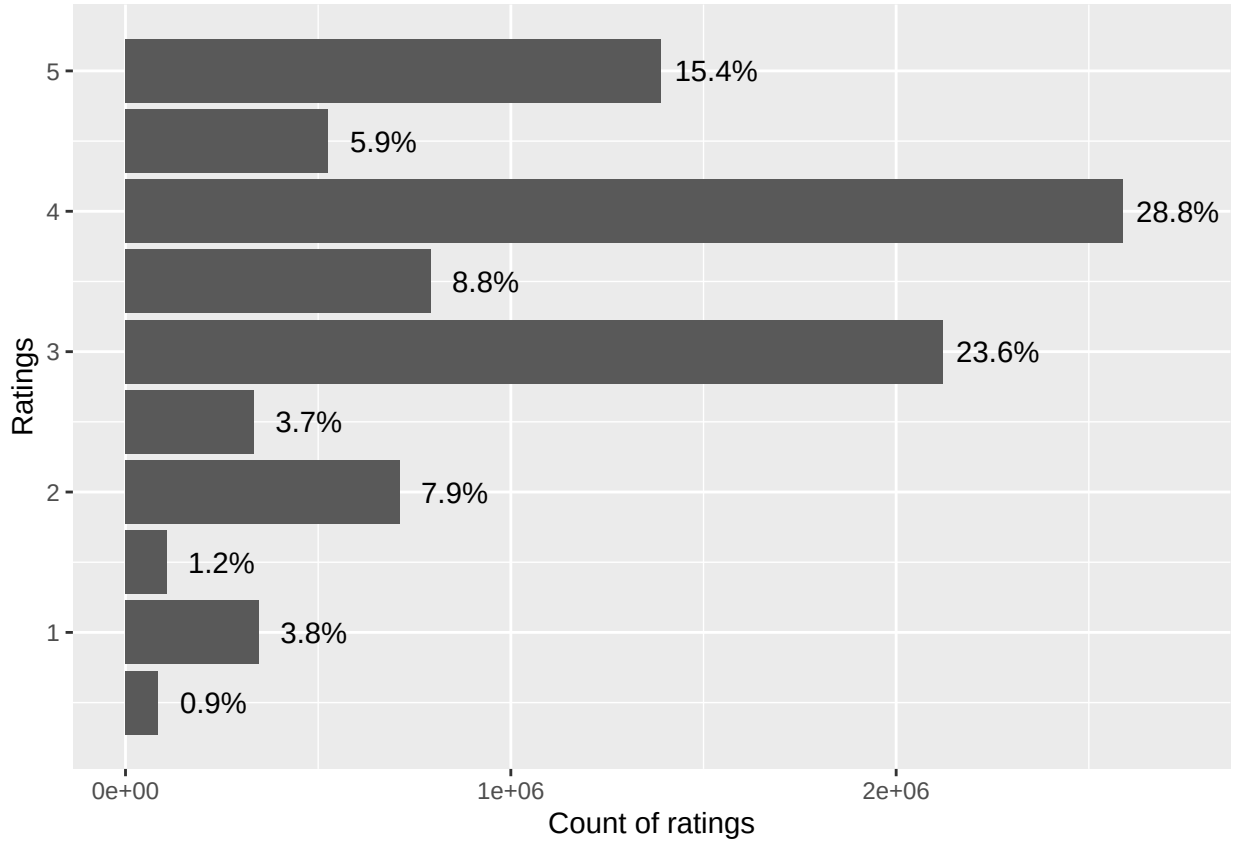

Figure 2: Histogram of ratings

Next, the frequency of ratings per movie was examined. A few movies were found to have received many ratings, with as many as 31,283 ratings for the most-rated movie. Conversely, many movies received few ratings, with as few as 1 rating for the least-rated movie. In fact, 125 movies were rated only once. Furthermore, 2,685 movies were found to be in the bottom 25th percentile in number of ratings received (30 ratings or fewer). The mean number of ratings per movie is 843.

Unsurprisingly, the movies that received the most ratings are also popular releases:

```
# Most rated movies
edx_train |>
  count(movieId, title, name = "n_ratings", sort = TRUE) |>
  head(10L) |>
  kable()
```

movieId	title	n_ratings
296	Pulp Fiction	31283
356	Forrest Gump	31024
593	Silence of the Lambs, The	30393
480	Jurassic Park	29369
318	Shawshank Redemption, The	28058
110	Braveheart	26263
589	Terminator 2: Judgment Day	26068
457	Fugitive, The	26016
260	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars)	25764
150	Apollo 13	24359

Conversely, those that received the fewest ratings are obscure releases:

```
# Least rated movies
edx_train |>
  count(movieId, title, name = "n_ratings", sort = TRUE) |>
  tail(10L) |>
  kable()
```

	movieId	title	n_ratings
10668	64754	Love	1
10669	64897	Mr. Wu	1
10670	64926	Battle of Russia, The (Why We Fight, 5)	1
10671	64944	Face of a Fugitive	1
10672	64953	Dirty Dozen, The: The Fatal Mission	1
10673	64976	Hexed	1
10674	65006	Impulse	1
10675	65011	Zona Zamfirova	1
10676	65025	Double Dynamite	1
10677	65027	Death Kiss, The	1

Figure 5 shows the number of movies that have received a given number of ratings. It confirms that some movies receive few ratings (left side of the chart) while few movies receive many ratings (right tail).

```
# Frequency of ratings
edx_train |>
  count(movieId, name = "ratings_per_movie") |>
  count(ratings_per_movie, name = "frequency") |>
```

```
  ggplot(mapping = aes(x = ratings_per_movie, y = frequency)) +
  geom_vline(xintercept = nrow(edx_train) / n_distinct(edx_train$movieId)) +
  annotate(
    geom  = "text",
    label = format(round(nrow(edx_train) / n_distinct(edx_train$movieId), 0), nsmall = 0),
    x     = nrow(edx_train) / n_distinct(edx_train$movieId) - 150,
    y     = -10
  ) +
  geom_point() +
  scale_x_log10() +
  labs(x = "Count of ratings per movie", y = "Count of movies")
```

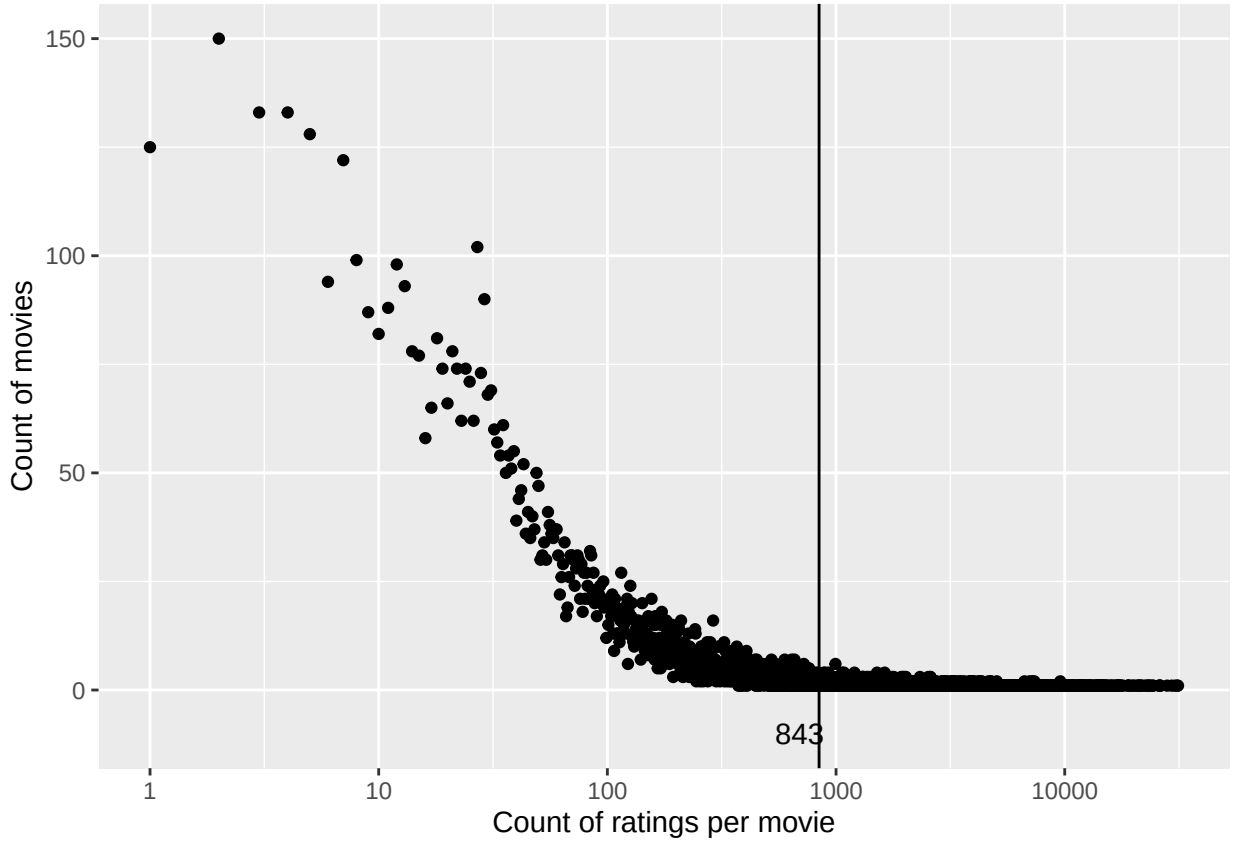

Figure 3: Scatterplot of number of ratings per movie

The low number of ratings received by some movies decreases the confidence in the reliability of those ratings. To illustrate, the following list of best-rated movies shows that they are obscure releases with very few ratings:

```
# Highest-rated movies (top n)
# We find that highest-rated movies also have very few (1-4) ratings each
edx_train |>
  group_by(movieId, title) |>
  summarize(mean_rating = mean(rating), n_rating = n()) |>
  arrange(desc(mean_rating)) |>
  head(10L) |>
  kable()
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
```

```
## `.groups` argument.
```

movieId	title	mean_rating	n_rating
5194	Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva)	5.000000	3
33264	Satan's Tango (Sátántangó)	5.000000	2
42783	Shadows of Forgotten Ancestors	5.000000	1
51209	Fighting Elegy (Kenka erejii)	5.000000	1
53355	Sun Alley (Sonnenallee)	5.000000	1
64275	Blue Light, The (Das Blaue Licht)	5.000000	1
65001	Constantine's Sword	4.750000	2
4454	More	4.714286	7
5849	I'm Starting From Three (Ricomincio da Tre)	4.666667	3
26048	Human Condition II, The (Ningen no joken II)	4.666667	3

If we remove the bottom 25th percentile of movies by number of ratings, the same analysis returns a more plausible set of best-rated movies:

```
# Same, but now excluding the 25th bottom percentile in # of ratings
# This returns a totally different set (zero overlap)
edx_train |>
  group_by(movieId, title) |>
  summarize(mean_rating = mean(rating), n_rating = n()) |>
  arrange(desc(mean_rating)) |>
  filter(n_rating > 30L) |>
  head(10L) |>
  kable()
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
```

movieId	title	mean_rating	n_rating
318	Shawshank Redemption, The	4.460101	28058
858	Godfather, The	4.413570	17731
50	Usual Suspects, The	4.366210	21616
527	Schindler's List	4.365098	23228
922	Sunset Blvd. (a.k.a. Sunset Boulevard)	4.325902	2938
6896	Shoah	4.325581	86
904	Rear Window	4.319451	7948
912	Casablanca	4.317117	11264
2019	Seven Samurai (Shichinin no samurai)	4.316496	5177
1212	Third Man, The	4.315529	2930

Likewise, the following list of worst-rated movies also shows obscure releases with very few ratings:

```
# Lowest-rated movies (bottom n)
# We find that lowest-rated movies also have very few (1-199) ratings each
edx_train |>
  group_by(movieId, title) |>
  summarize(mean_rating = mean(rating), n_rating = n()) |>
  arrange(desc(mean_rating)) |>
  tail(10L) |>
  kable()
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
```

movieId	title	mean_rating	n_rating
6189	Dischord	1.0000000	1
55324	Relative Strangers	1.0000000	1
6483	From Justin to Kelly	0.9322917	192
61348	Disaster Movie	0.9054054	37
7282	Hip Hop Witch, Da	0.8214286	14
8859	SuperBabies: Baby Geniuses 2	0.8070175	57
64999	War of the Worlds 2: The Next Wave	0.6666667	3
5805	Besotted	0.5000000	1
8394	Hi-Line, The	0.5000000	1
61768	Accused (Anklaget)	0.5000000	1

And again, temporarily removing the 25th percentile of least-rated movies returns a somewhat different set of worst-rated movies. Note that the *title* variable was dropped from the training dataset to free up memory and considering that it does not constitute a predictor variable.

```
# Same, but now excluding the 25th bottom percentile in # of ratings
# This returns a fairly different set (only three movies overlap)
edx_train |>
  group_by(movieId, title) |>
  summarize(mean_rating = mean(rating), n_rating = n()) |>
  arrange(desc(mean_rating)) |>
  filter(n_rating > 30L) |>
  tail(10L) |>
  kable()
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
```

movieId	title	mean_rating	n_rating
6587	Gigli	1.1853035	313
5672	Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie)	1.1714286	210
1826	Barney's Great Adventure	1.1650943	212
4775	Glitter	1.1622024	336
3574	Carnosaur 3: Primal Species	1.1418919	74
3027	Slaughterhouse 2	1.1363636	33
6371	Pokémon Heroes	1.0034014	147
6483	From Justin to Kelly	0.9322917	192
61348	Disaster Movie	0.9054054	37
8859	SuperBabies: Baby Geniuses 2	0.8070175	57

```
# Drop the title column
edx_train <- edx_train |>
  select(!title)
```

These observations are confirmed by plotting the average rating for every count of movie ratings. Figure 5 shows that movies with few ratings (those near the vertical axis) have a much wider rating (vertical) spread than those with many ratings, which tend to stabilize around the mode of 4.

```r
# Plot the relationship between rating count and rating score
edx_train |>
  group_by(movieId) |>
  summarize(mean_rating = mean(rating), n_rating = n()) |>
  ggplot(mapping = aes(x = n_rating, y = mean_rating)) +
  geom_point() +
  labs(x = "Count of ratings per movie", y = "Mean rating")
```

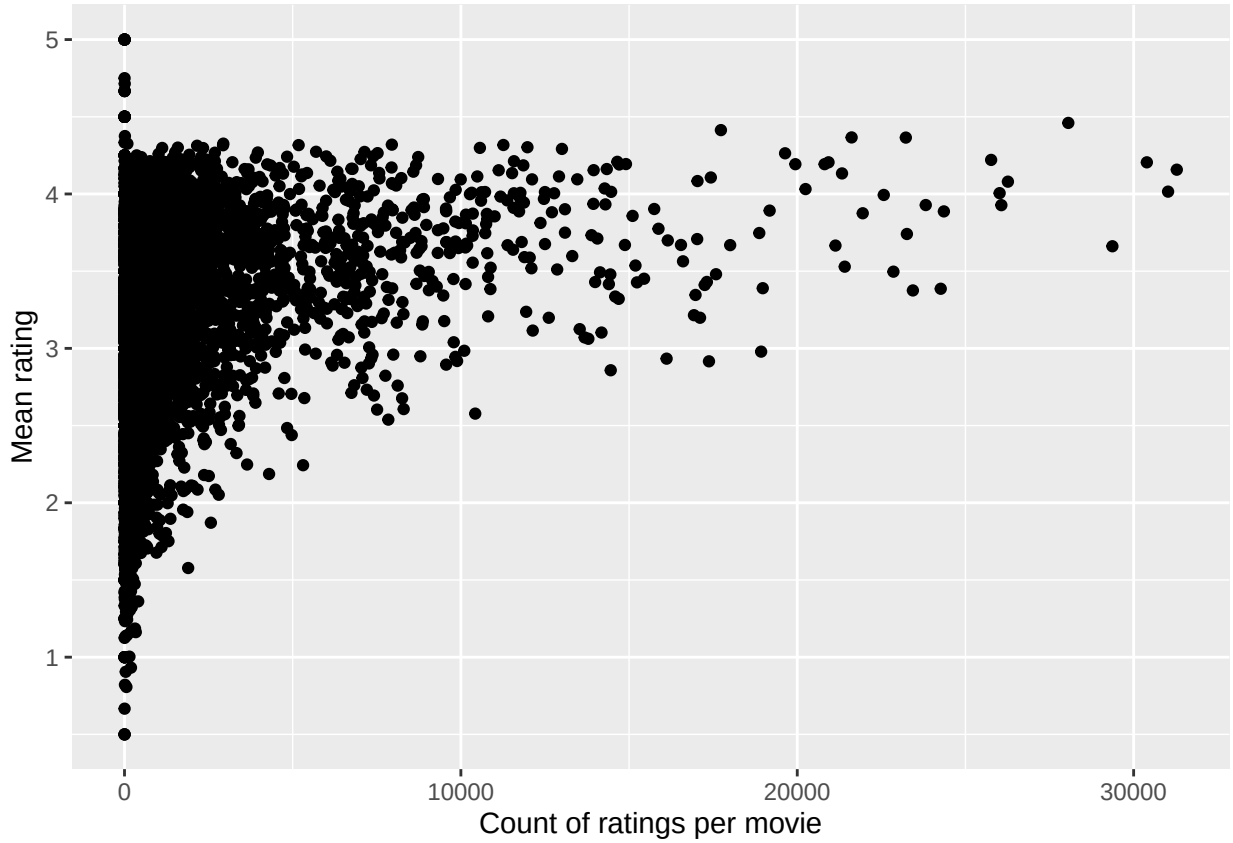

Figure 4: Scatterplot of mean rating per rating count

Lastly, a similar discrepancy in ratings was observed across users. Figure 5 plots the mean rating assigned by every user (on the vertical axis) against the number of times those users have given ratings. Many users who contributed very few ratings display a wider (vertical) spread in their ratings, relative to frequent reviewers who tend to assign that are, on average, close to the overall average of 3.51.

```r
# Ratings by reviewer
edx_train |>
  group_by(userId) |>
  summarize(mean_rating = mean(rating), n_rating = n()) |>
  ggplot(mapping = aes(x = n_rating, y = mean_rating)) +
  geom_point() +
  labs(x = "Count of ratings per user", y = "Mean rating")
```

In conclusion for the exploratory analysis of the ratings, we can state that an ideal prediction model will need to account for:

- A *movie effect* where certain movies are inherently rated higher than others;

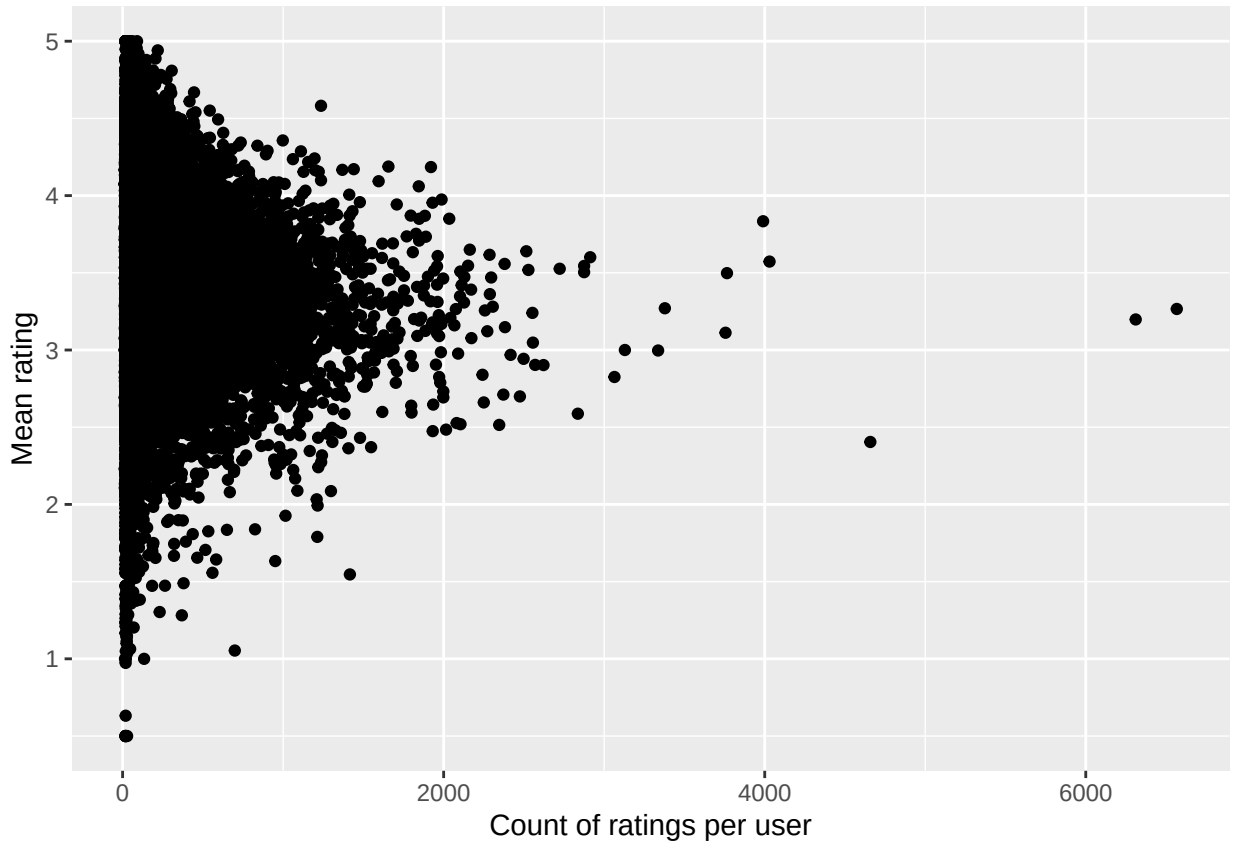

Figure 5: Scatterplot of mean rating per user

- A *number of ratings per movie effect* where the predictive power of ratings increases with their frequency;
- A *user effect* where certain users rate movies consistently higher than others.

timestamp

An integer representation of the Unix timestamp of when the review was made by the user, expressed in second elapsed since 00:00:00 UTC on Thursday, 1 January 1970. The earliest movie rating was assigned on 1995-01-09 and the latest on 2009-01-05.

Because the year of the rating might be a predictor of the rating, the *timestamp* variable was converted to a human-readable date and stored as a new date variable using the following code:

```
# Convert timestamp
edx_train <- edx_train |>
  mutate(
    year_rating = as.integer(
      as.POSIXlt(
        timestamp, origin = "1970-01-01"
      )$year + 1900L
    ),
    timestamp   = NULL
  )
```

Following these transformations, the edx_train dataset now contains one response variable, *rating* and 5 predictor variables, which are *movieId*, *userId*, *year_movie*, *year_rating*, and one *genre* binary variable for every cinematographic genre.

Manual predictions

Next, manual predictions were performed using several of the predictor variables, and their respective performance was measured using the root mean square error (RMSE). The lower the RMSE, the better. The target RMSE for this assignment is 0.8649 or lower.

First, a naive prediction model was created using just the mean of all ratings in the training set as the predicted response variable *rating*. This prediction was then compared to the actual values from the *final_holdout_test* set.

```
# Mean of all ratings in the training set
mu <- mean(edx_train$rating)

# RMSE without accounting for any effect
rmses <- c(
  rmses,
  naive = RMSE(pred = mu, obs = final_holdout_test$rating)
)
```

The resulting RMSE of 1.0603341 forms the starting point from which we must improve the predictive power of the model.

Next, a model accounting for the bias in individual user ratings was created:

```
# Measure the effect relative to the mean of all ratings
user_bias <- edx_train |>
  group_by(userId) |>
  summarize(user_bias = mean(rating - mu))

# Create a set of predicted ratings with the same cardinality as the test set
user_pred <- final_holdout_test |>
```

```
  left_join(
    user_bias,
    by = "userId"
  ) |>
  mutate(pred = mu + user_bias) |>
  select(userId, pred)

# RMSE after accounting for user effect
rmses <- c(
  rmses,
  user = RMSE(pred = user_pred$pred, obs = final_holdout_test$rating)
)
```

The resulting RMSE of 0.9779362 is a stark improvement over the naive baseline.

Next, a model accounting for the bias in individual movie ratings was created in similar fashion:

```
# Measure the effect relative to the mean of all ratings
movie_bias <- edx_train |>
  group_by(movieId) |>
  summarize(movie_bias = mean(rating - mu))

# Create a set of predicted ratings with the same cardinality as the test set
movie_pred <- final_holdout_test |>
  left_join(
    movie_bias,
    by = "movieId"
  ) |>
  mutate(pred = mu + movie_bias) |>
  select(movieId, pred)

# RMSE after accounting for movie effect
rmses <- c(
  rmses,
  movie = RMSE(pred = movie_pred$pred, obs = final_holdout_test$rating)
)
```

The resulting RMSE of 0.944011 is a further improvement toward minimizing the RMSE, but still above the goal.

The next step was to create a similar model to account for a bias in individual cinematographic genres. The single, concatenated string of genres assigned to each movie in the original training dataset was used for this purpose.

```
# Measure the effect relative to the mean of all ratings
multi_genre_bias <- edx_train |>
  group_by(genres) |>
  summarize(multi_genre_bias = mean(rating - mu))

# Create a set of predicted ratings with the same cardinality as the test set
multi_genre_pred <- final_holdout_test |>
  left_join(
    multi_genre_bias,
    by = "genres"
  ) |>
  mutate(pred = mu + replace_na(multi_genre_bias, 0)) |>
```

```
  select(genres, pred)

# RMSE after accounting for multi-genre effect
rmses <- c(
  rmses,
  multi_genre = RMSE(pred = multi_genre_pred$pred, obs = final_holdout_test$rating)
)
```

The resulting RMSE of 1.0181863 is worse, which indicates that concatenated *genres* may not be a useful predictor for our model. To confirm that, the concatenated *genres* were split into as many rows as there are unique genres in the string, so that the bias of each genre relative to the mean of all ratings could be accounted for.

```
# Split the genres column and pivot it into rows in the training set
edx_train_longer <- edx_train |>
  select(genres, rating) |>
  separate_longer_delim(genres, delim = "|")

# Measure the effect relative to the mean of all ratings
single_genre_bias <- edx_train_longer |>
  group_by(genres) |>
  summarize(single_genre_bias = mean(rating - mu))

# Split the genres column and pivot it into rows in the test set
final_holdout_test_longer <- final_holdout_test |>
  separate_longer_delim(genres, delim = "|")

# Create a set of predicted ratings w/ the same cardinality as the long test set
single_genre_pred <- final_holdout_test_longer |>
  left_join(
    single_genre_bias,
    by = "genres"
  ) |>
  mutate(pred = mu + single_genre_bias) |>
  select(genres, pred)

# RMSE after accounting for single-genre effect
rmses <- c(
  rmses,
  single_genre = RMSE(
    pred = single_genre_pred$pred,
    obs = final_holdout_test_longer$rating
  )
)
```

The resulting RMSE of 1.0454753 is still better than the naive model, but worse than that of treating concatenated genres as one.

Next, a possible rating bias related to the year of movie release was tested.

```
# Summarize the effect relative to the mean of all ratings
year_movie_bias <- edx_train |>
  group_by(year_movie) |>
  summarize(year_movie_bias = mean(rating - mu))

# Create the year_movie variable in the test set as well so that we can join
```

```r
final_holdout_test <- final_holdout_test |>
  mutate(
    year_movie  = as.integer(str_sub(title, start = -5L, end = -2L)),
    title       = str_sub(title, end = -8L)
  )

# Create a set of predicted ratings with the same cardinality as the test set
year_movie_pred <- final_holdout_test |>
  left_join(
    year_movie_bias,
    by = "year_movie"
  ) |>
  mutate(pred = mu + year_movie_bias) |>
  select(year_movie, pred)

# RMSE after accounting for year of release effect
rmses <- c(
  rmses,
  year_movie = RMSE(pred = year_movie_pred$pred, obs = final_holdout_test$rating)
)
```

The resulting RMSE of 1.0494642 is also marginally better than the naive prediction, but not an improvement over the *movieId* and *userId* effect models.

Lastly, a similar model was created based on the year of the rating.

```r
# Summarize the effect relative to the mean of all ratings
year_rating_bias <- edx_train |>
  group_by(year_rating) |>
  summarize(year_rating_bias = mean(rating - mu))

# Create the year_rating variable in the test set as well so that we can join
final_holdout_test <- final_holdout_test |>
  mutate(
    year_rating = as.integer(as.POSIXlt(timestamp, origin = "1970-01-01")$year + 1900L),
    timestamp   = NULL
  )

# Create a set of predicted ratings with the same cardinality as the test set
year_rating_pred <- final_holdout_test |>
  left_join(
    year_rating_bias,
    by = "year_rating"
  ) |>
  mutate(pred = mu + year_rating_bias) |>
  select(year_rating, pred)

# RMSE after accounting for year of release effect
rmses <- c(
  rmses,
  year_rating = RMSE(pred = year_rating_pred$pred, obs = final_holdout_test$rating)
)
```

The resulting RMSE of 1.0586224 is the worst of all the manual, single-predictor models so far. Can a model combining all these individual predictors perform better? To answer this question, a combined model was

created using the following code:

```
# Create a set of predicted ratings with the same cardinality as the test set
movieusergenreyear_pred <- final_holdout_test |>
  left_join(movie_bias,       by = "movieId") |>
  left_join(user_bias,        by = "userId") |>
  left_join(multi_genre_bias, by = "genres") |>
  left_join(year_movie_bias,  by = "year_movie") |>
  left_join(year_rating_bias, by = "year_rating") |>
  mutate(
    pred = mu + movie_bias + user_bias + multi_genre_bias +
      year_movie_bias + year_rating_bias) |>
  select(movieId, userId, genres, year_movie, year_rating, pred)

# RMSE after accounting for movie, user, genre, and year effects
rmses <- c(
  rmses,
  movieusergenreyear = RMSE(
    pred = movieusergenreyear_pred$pred,
    obs  = final_holdout_test$rating)
)
```

The resulting RMSE of 0.9707198 is back under unity, but still worse than the best-performing model so far, which accounts only for the movie effect. Next, the *year_movie* and *year_rating* predictors were removed to test a simpler combined model:

```
# Create a set of predicted ratings with the same cardinality as the test set
movieusergenre_pred <- final_holdout_test |>
  left_join(movie_bias,       by = "movieId") |>
  left_join(user_bias,        by = "userId") |>
  left_join(multi_genre_bias, by = "genres") |>
  mutate(pred = mu + movie_bias + user_bias + multi_genre_bias) |>
  select(movieId, userId, genres, pred)

# RMSE after accounting for movie, user, and genre effects
rmses <- c(
  rmses,
  movieusergenre = RMSE(
    pred = movieusergenre_pred$pred,
    obs  = final_holdout_test$rating)
)
```

The resulting RMSE of 0.9455764 is further improved and almost on par with the simple movie effect model. Can we do better by using only *movieId* and *userId* as predictors?

```
# Create a set of predicted ratings with the same cardinality as the test set
movieuser_pred <- final_holdout_test |>
  left_join(movie_bias, by = "movieId") |>
  left_join(user_bias,  by = "userId") |>
  mutate(pred = mu + movie_bias + user_bias) |>
  select(movieId, userId, pred)

# RMSE after accounting for movie and user effects
rmses <- c(
  rmses,
  movieuser = RMSE(
```

```
    pred = movieuser_pred$pred,
    obs  = final_holdout_test$rating)
)
```

The resulting RMSE of 0.8853043 shows that the minimized RMSE among all manual models is achieved by accounting for only biases in individual movie and user ratings combined. However, it is still above the target RMSE of 0.8649. To improve the RMSE further, matrix factorization will be required.

Recosys model

recosystem is a recommender system (available as an R package) based on the *LIBMF* open-source matrix factorization library. It takes a sparse matrix of users (in rows), items (in columns), and ratings (in cells), and predicts missing rating values in blank cells based on user and item biases. The package starts by training the model against the dataset, then tunes the parameters to minimize the RMSE, then exports the model, and finally predicts rating values against an independent, never-seen-before testing dataset.

The *recosystem* package was chosen because it uses exactly two predictor variables (i and j) that mirror the user-movie effect model that was shown earlier to minimize the RMSE.

To start, two data sources for the training and testing datasets were created from sparse matrices in triplet form comprised of users (i), movies (j), and ratings (x).

```
edx_train_datasource <- data_matrix(
  mat = sparseMatrix(
    i    = edx_train$userId,
    j    = edx_train$movieId,
    x    = edx_train$rating,
    repr = "T"
  )
)

final_holdout_test_datasource <- data_matrix(
  mat = sparseMatrix(
    i    = final_holdout_test$userId,
    j    = final_holdout_test$movieId,
    x    = final_holdout_test$rating,
    repr = "T"
  )
)
```

Next, a *recosystem* object was instantiated with training parameters that include 50 latent factors, a 10-fold cross-validation, and the use of multiple CPU threads to speed up runtime.

```
# Instantiate the RecoSys object
r <- Reco()

# Set the training parameters
opts_tune <- list(
  dim      = 200L, # Latent factors
  nbin     = 30L,  # Number of bins
  nfold    = 10L,  # k-fold cross-validation
  nthread  = 20L   # CPU threads
)

# Tune the model to the optimal parameters
opts_train <- r$tune(train_data = edx_train_datasource, opts = opts_tune)
```

```
## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
```

```
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)
```

```
## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)

## Warning in r$tune(train_data = edx_train_datasource, opts = opts_tune):
## insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1 blocks
## is suggested)
```

The model was tuned automatically and the following tuning parameters were returned:

```r
# Display the optimal parameters
opts_train$min
```

```
## $dim
## [1] 200
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.01
##
## $costq_l1
## [1] 0
##
## $costq_l2
## [1] 0.1
##
## $lrate
## [1] 0.1
##
## $loss_fun
## [1] 0.7816039
```

Next, the model was trained against the *edx_training* data source using those tuning parameters:

```r
# Train the model with tuned parameters
r$train(
  edx_train_datasource,
  opts = c(opts_train$min, nthread = 20L, nbin = 30L)
)
```

```
## Warning in r$train(edx_train_datasource, opts = c(opts_train$min, nthread =
## 20L, : insufficient blocks may slow down the trainingprocess (4*nr_threads^2+1
## blocks is suggested)
```

Lastly, the trained model was used to predict ratings on the *final_holdout_test* set, and the RMSE was calculated.

```
# RMSE from the recosys package
rmses <- c(
  rmses,
  recosys = RMSE(
    pred = r$predict(test_data = final_holdout_test_datasource),
    obs  = final_holdout_test$rating)
)
```

The resulting RMSE of 0.7739731 outperforms the earlier user-movie effect model thanks to the added matrix factorization. It is also significantly lower than the target RMSE of 0.8649.

Results

The script completed in 1.05445255166954.

Conclusion