Movielens project submission

Emelie Wahlstedt

2024-06-01

Introduction

This project aims to create a movie recommendation system based on a dataset provided as part of the Data Science: Capstone module, which is the final part of the HarvardX Data Science Professional Certificate programme. The dataset provided for this analysis, “movielens”, contains a large list of movies that have been rated by different users - but not all movies have been rated by all users. The goal of this project is to build a recommendation system that is able to fill in these gaps - essentially, what would user x think about movie y if they watched it?

The “movielens” dataset will be divided into a test set (“edx”) and a final holdout test set (“final_holdout_test) through code provided as part of the instructions for the project.

In this report, we explore the dataset, modify the dataset to facilitate modelling and add predictors that may be useful, and test a number of different approaches to creating a model for a recommendation system.

The following libraries have been used for the project: library(caret) | library(data.table) | library(dplyr) | library(FNN) | library(glmnet) | library(ggrepel) | library(h2o) | library(knitr) | library(lubridate) | library(speedglm) | library(tidyverse) | library(xgboost) |

Method/Analysis

The first thing to be done is to create train and test datasets as instructed for this project, using the code provided in the course documentation. Once this is complete, exploration and data cleaning can begin.

Data exploration and tidying of data

The datasets edx and final_holdout_test have now been created. A look at the first few rows of the edx dataset will be useful:

```
##   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
```

There are six columns in the dataset, with the rating column being the target variable and the other five being potential predictors.

The “timestamp” variable which is the “Unix” or “Epoch” format which, while it can be read by R without issue, is difficult for most humans to interpret. It is also unnecessarily precise, which may make things difficult down the line. The timestamp variable is rounded to the nearest month (YYYY-MM-01), but keep it in numeric format:

```r
edx$timestamp <- as.numeric(round_date(as_datetime(edx$timestamp), "month"))
```

The column “titles” includes the year the movie was released, at the end of the movie title in brackets. The release year could be a predictor, so it is separated out into its own column. The resulting dataset is named edx_tidy:

```r
edx_tidy <- edx %>% separate(title, c("title", "release_year"), -6) # Get the release
 ↪  year, but it's still in brackets

release_year_new <- str_extract(edx_tidy$release_year, "\\d{4}") # Remove brackets

edx_tidy$release_year <- as.numeric(release_year_new) # Overwrite the column in main
 ↪  dataset with the one without brackets

rm(release_year_new) # Remove as no longer needed
```

Looking at the other variables, “userId”, “movieId”, and “genres” seem to logically be able to give some information on how a particular movie would be rated by a random user - different users probably have different preferences, and different movies may have more or less wide appeals. Some genres are probably more popular than others. Hence, these may be important as predictors in the recommendation system model that wil lbe the end result of this project. There may also be some effect of time - for example, if a movie is very new, it may not have received as many ratings yet, because very few users have watched and rated it at this point. This could be significant for the average rating that the movie obtains.

Let’s have a closer look at the summary statistics of the dataset:

Table 1: Overview of the edx dataset

Statistic	Value
Number of ratings	9000055
Number of unique movies	10677
Number of unique users	69878
Number of unique genre combinations	797

Statistic	Value
Mean of ratings	4
Median of ratings	4
Mode of ratings	4

This is a very large dataset - over 9 million rows - which will need to be taken into account when selecting a model, as unlimited computational power is not available. Rather, a very average laptop is.

The data quality is another thing to check before attempting to build a model - are there any missing values in the edx and final_holdout_test datasets?

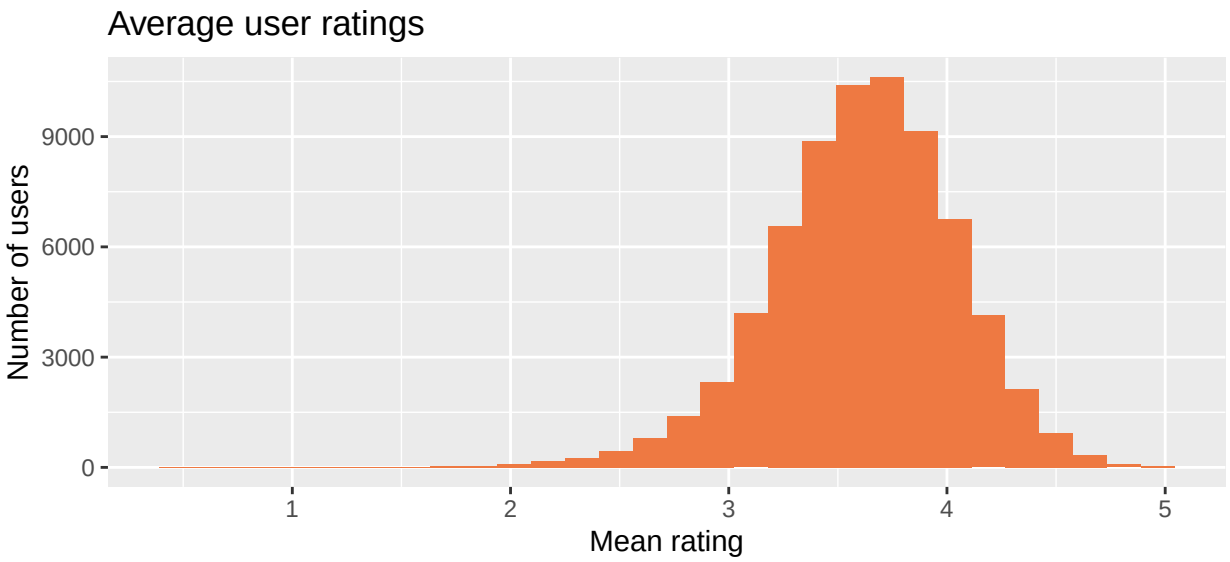
Dataset	Missing_values
edx	0
final_holdout_test	0

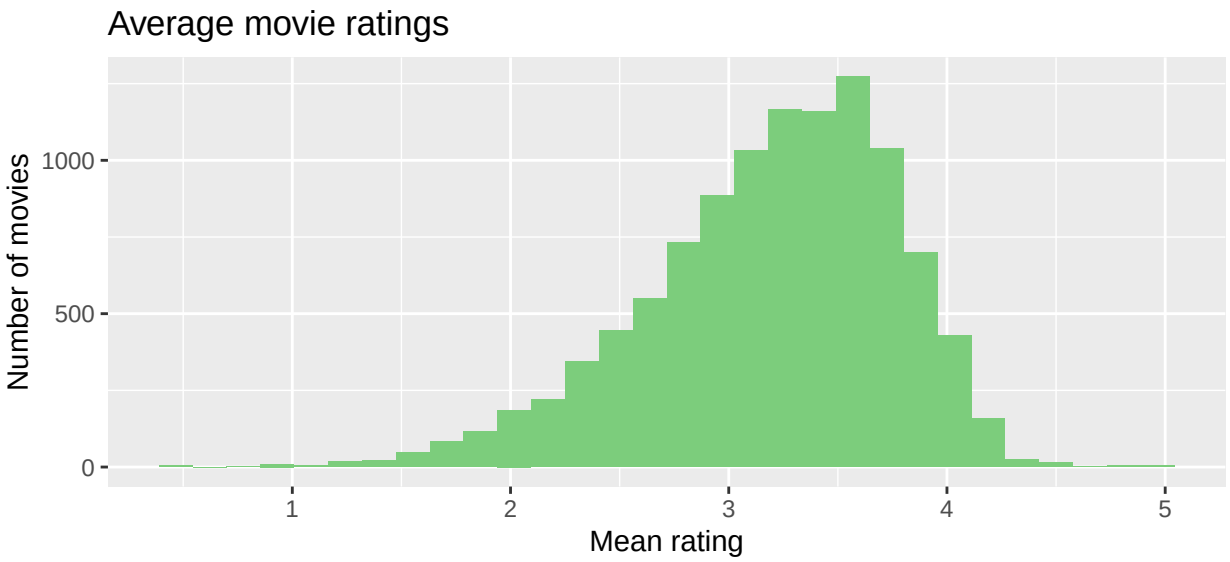
No missing values, great!

MovieId and UserId

It may be informative to plot some of the data to get an idea of the distribution of some of the variables. Average ratings are used for some plots as plotting the entire dataset will be too visually cluttered and will take a long time.

First, let's look at the average ratings for each userId and movieId:

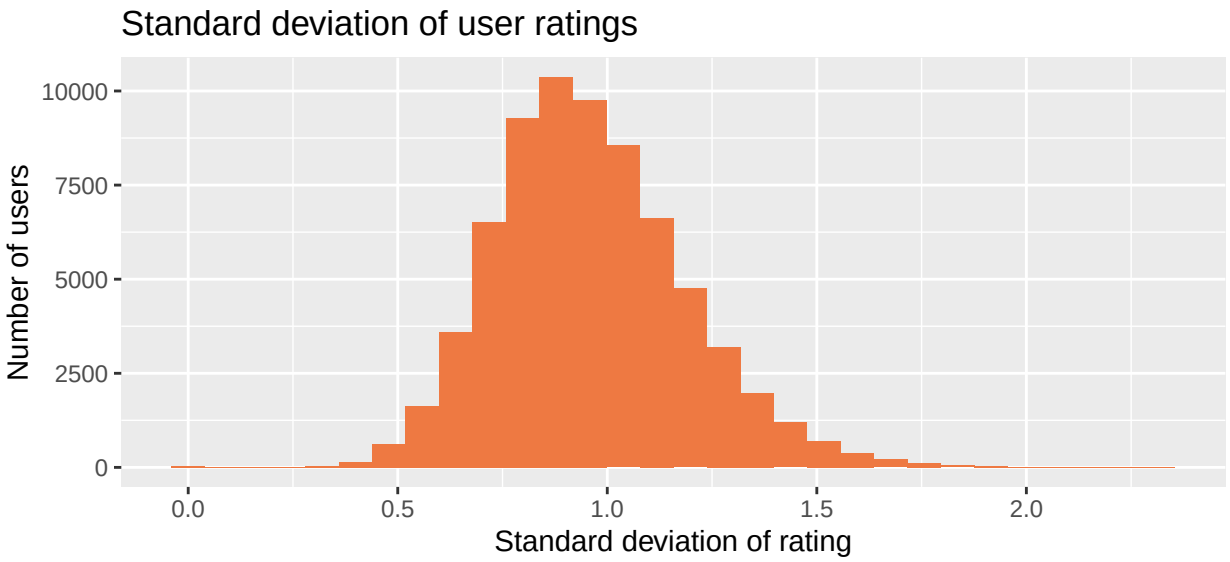


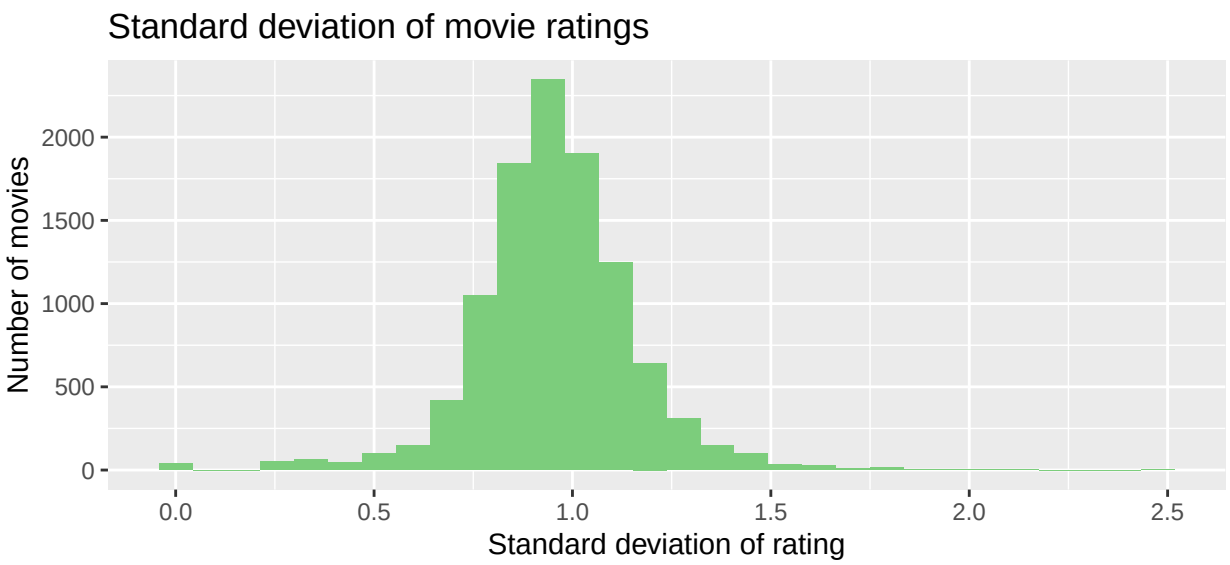


Movies in this dataset certainly have average ratings that differ - some are considered good and others bad, ad many are somewhere in the middle. The average rating of a movie may be a useful predictor. The number of times a movie has been rated may also be interesting for making predictions, as movies with more ratings seem to have slightly higher mean ratings.

Similarly, there are users who are generally positive and those who are less so - this again may be useful for making predictions. The number or ratings per user also seems to have some impact on the user's average rating, but the data is more evenly distributed than the movie average ratings. Nevertheless, in combination with other predictors the number of times a user has rated a movie may still be a useful predictor.

If average ratings can be a useful predictor, perhaps standard deviation can be as well - let's plot this and see:





The standard deviations also vary across userIds and movieIds - more so for userIds than movieIds. There is still some variation meaning these variables could be useful for predictions.

To capture the above, edx_wide is updated with new predictors. For the standard deviations, movies rated only once or users that have only rated one movie may come out as NA, and therefore these are set to zero:

```
edx_tidy <- edx_tidy %>%
  group_by(userId) %>% # Create variables by userId
  mutate(user_avg_rating = mean(rating),
         user_rating_count = n(),
         user_sd_rating = sd(rating)) %>%
```

```
  ungroup() %>%
  group_by(movieId) %>% # Create varables by movieId
  mutate(movie_avg_rating = mean(rating),
         movie_rating_count = n(),
         movie_sd_rating = sd(rating)) %>%
  ungroup()

edx_tidy$movie_sd_rating[is.na(edx_tidy$movie_sd_rating)] <- 0 # Fill any missing values
 ↪  with 0 (movies with only one rating have no SD)
edx_tidy$user_sd_rating[is.na(edx_tidy$user_sd_rating)] <- 0 # Fill any missing values
 ↪  with 0 (users that have rated only one movie have no SD)

setDT(edx_tidy) # Back to data.table as above operations may convert to data.frame
```

The dataset is large - therefore the format of data.frame makes analysis and modelling slow. Changing the format to data.table, as done in at the end of the previous code chunk, significantly improves processing speed and avoids setting ablaze the very average laptop in use for this project.

Movie Genres

The genre column in the dataset allows for several genres to be listed for each movie - but how many individual genres are present in the dataset and how many movies fall under each genre? To find this out, the genre column is split into separate columns, each named after the genres they represent. In order to do this a dataset called edx_wide is created:

```
# Separate rows by genres
edx_expanded <- edx_tidy[, .(genre = unlist(strsplit(as.character(genres), "\\|"))),
                   by = .(userId, movieId, release_year, title, rating, timestamp,
 ↪  user_avg_rating, user_rating_count, user_sd_rating, movie_avg_rating,
 ↪  movie_rating_count, movie_sd_rating)]

# Create genre_indicator column
```

```r
edx_expanded[, genre_indicator := 1]

# Pivot to wide format
edx_wide <- dcast(edx_expanded, userId + movieId + title + release_year + rating +
 ↪  timestamp + user_avg_rating + user_rating_count + user_sd_rating + movie_avg_rating +
 ↪  movie_rating_count + movie_sd_rating ~ genre,
                  value.var = "genre_indicator", fill = 0)

#rm(edx_expanded) # Remove as no longer needed

setDT(edx_wide) # Set to data.table again as some of the operations above may have
 ↪  converted the data back into a data.frame

# Select unique movies by movieId
unique_movies <- unique(edx_wide, by = "movieId")

# Select only the genre columns
genre_columns <- setdiff(names(unique_movies), c("userId", "movieId", "title",
 ↪  "release_year", "rating", "timestamp", "user_avg_rating", "user_rating_count",
 ↪  "user_sd_rating", "movie_avg_rating", "movie_rating_count", "movie_sd_rating"))

# Sum occurrences of each genre across unique movies
Count <- colSums(unique_movies[, .SD, .SDcols = genre_columns])

Count <- data.frame(Count)
Genre <- rownames(Count)
genres <- cbind(Genre, data.frame(Count, row.names=NULL))
genres <- genres %>% arrange(desc(Count))
```

Table 3: Number of movies within each genre (most movies appear in more than one genre)

Genre	Count
Drama	5336
Comedy	3703
Thriller	1705
Romance	1685
Action	1473
Crime	1117
Adventure	1025
Horror	1013
Sci-Fi	754
Fantasy	543
Children	528
War	510
Mystery	509
Documentary	481
Musical	436
Animation	286
Western	275
Film-Noir	148
IMAX	29
(no genres listed)	1

There are 20 genre labels, with one of these being “(no genres listed)” - only one movie is in this category. If this is a very popular movie with many ratings, it may impact on any model that is run, so taking a further look may be useful.

```
##    userId movieId rating  timestamp                title             genres
##     <int>   <int>  <num>      <num>               <char>             <char>
## 1:   7701    8606    5.0 1191196800 Pull My Daisy (1958) (no genres listed)
```

```
## 2:  10680    8606    4.5 1170288000 Pull My Daisy (1958) (no genres listed)
## 3:  29097    8606    2.0 1088640000 Pull My Daisy (1958) (no genres listed)
## 4:  46142    8606    3.5 1225497600 Pull My Daisy (1958) (no genres listed)
## 5:  57696    8606    4.5 1230768000 Pull My Daisy (1958) (no genres listed)
## 6:  64411    8606    3.5 1096588800 Pull My Daisy (1958) (no genres listed)
## 7:  67385    8606    2.5 1188604800 Pull My Daisy (1958) (no genres listed)
```

This movie has only been rated seven times, so it won't make a huge difference to the analysis and can be left in the dataset for now.

Some of the genres are named in a way that is not helpful for further analysis, and therefore these are renamed. It will also be helpful if all the genre columns are labeled as such:

```
# Add prefix 'genre_' to all genre columns
genre_columns <- setdiff(names(edx_wide), c("userId", "movieId", "title", "release_year",
↪  "rating", "timestamp", "user_avg_rating", "user_rating_count", "user_sd_rating",
↪  "movie_avg_rating", "movie_rating_count", "movie_sd_rating"))
new_genre_columns <- paste("genre", genre_columns, sep = "_")
edx_wide <- setnames(edx_wide, old = genre_columns, new = new_genre_columns)

# Change problematic column names
colnames(edx_wide)[which(names(edx_wide) == "genre_(no genres listed)")] <-
↪  "genre_no_genres_listed"
colnames(edx_wide)[which(names(edx_wide) == "genre_Film-Noir")] <- "genre_Film_Noir"
```

In effect there are 19 different genres, but these are in turn combined in the genres column to form different unique combinations - there are in fact 797 unique combinations of genres, which could each be considered a sub-genre of their own.

Making the dataset more lean - selecting predictors

There are a lot of predictors now - 30 different columns that could be used in the model. This seems like a lot, and since the computational power available is a very average laptop, this may not work. Some predictors will surely be more useful than others - this is investigated below.

First of all the title column can be removed, as this will not provide any information that the more concise predictor movieId cannot:

```
# Title column removed as will not be useful for analysis
edx_wide <- edx_wide[, !"title", with = FALSE]
```

It may be useful to check if any of the predictors are collinear:

```
# Check for Collinearity using Correlation Matrix
cor_matrix_wide <- cor(edx_wide[, !"rating", with = FALSE])
findCorrelation(cor_matrix_wide)
```

```
## integer(0)
```

There doesn't seem to be any collinearity, so all the predictors can be kept for now. To move further with the selection of predictors, using the variable importance feature of random forest models may be useful. To do this, the edx_wide dataset needs to be partitioned into train and test sets.

Creating train and test sets from the edx_wide dataset

The train and test sets are created in same way as the edx and final_holdout_test sets were created, with a 10% test set and 90% train set:

```
#### Creating train and test datasets from the edx_wide dataset

# Create train and test sets from edx_wide - use same methodology as used for creation of
↪  edx and final_holdout_test sets above
set.seed(1)

edx_test_index_wide <- createDataPartition(y = edx_wide$rating, times = 1, p = 0.1, list
↪  = FALSE)

edx_train_wide <- edx_wide[-edx_test_index_wide,]
```

```
temp <- edx_wide[edx_test_index_wide,]

# Make sure userId and movieId in final hold-out test set are also in edx_train set (to
↪  improve predictions as same movies are in both sets?)

edx_test_wide <- temp %>%
  semi_join(edx_train_wide, by = "movieId") %>%
  semi_join(edx_train_wide, by = "userId")

# Add rows removed from edx_test set back into edx_train set

removed <- anti_join(temp, edx_test_wide)

edx_train_wide <- rbind(edx_train_wide, removed)

rm(edx_test_index_wide, temp, removed) # Remove as not needed

setDT(edx_train_wide) # Covert to data.table format if not already
setDT(edx_test_wide) # Covert to data.table format if not already
```

Run random forest model to find importance of predictors

The train and test sets are now available for use. Using caret's train function to run a random forest model crashed R, so through some research a random forest model using h2o, which is able to run large datasets, was identified and is used here.

```
# Convert data to H2O frames
train_h2o <- as.h2o(edx_train_wide)
test_h2o <- as.h2o(edx_test_wide)

# Define the response and predictor variables
```

```
target <- "rating"
predictors <- setdiff(names(edx_train_wide), target)

# Train the random forest model using H2O
model <- h2o.randomForest(
  y = target,
  x = predictors,
  training_frame = train_h2o,
  ntrees = 100,
  mtries = 3,
  min_rows = 5,
  seed = 1
)

# Predict on the test set
predictions <- h2o.predict(model, test_h2o)

# Extract variable importance
importance <- h2o.varimp(model)
```

Table 4: The top ten most important predictors by importance

variable	relative_importance	scaled_importance	percentage
movie_avg_rating	66020052	1.0000000	0.3340442
user_avg_rating	61068584	0.9250005	0.3089911
movie_sd_rating	18752778	0.2840467	0.0948842
movie_rating_count	9811152	0.1486087	0.0496419
user_sd_rating	8028501	0.1216070	0.0406221
release_year	6935175	0.1050465	0.0350902
user_rating_count	4390950	0.0665093	0.0222171
genre_Drama	4097851	0.0620698	0.0207341
movieId	3448317	0.0522314	0.0174476

variable	relative_importance	scaled_importance	percentage
timestamp	3084827	0.0467256	0.0156084

The five most important variables are movie_avg_rating, user_avg_rating, movie_sd_rating, movie_rating_count and user_sd_rating. Five seems like a good number - let's go with these. The top two predictors appears to be one order of magnitude more important than the others - let's keep thisin mind too.

To use these for model testing, a dataset with only these predictors are created (userId and movieId are also included for readability, and of course rating as this is the target variable), and train and test datasets are altered in the same way:

```
# Remove columns from the full dataset
edx_final <- edx_wide[, .(rating, userId, movieId, movie_avg_rating, user_avg_rating,
 ↪  movie_sd_rating, movie_rating_count, user_sd_rating)]

# Remove columns from train and test sets
edx_train_final <- edx_train_wide[, .(rating, userId, movieId, movie_avg_rating,
 ↪  user_avg_rating, movie_sd_rating, movie_rating_count, user_sd_rating)]
edx_test_final <- edx_test_wide[, .(rating, userId, movieId, movie_avg_rating,
 ↪  user_avg_rating, movie_sd_rating, movie_rating_count, user_sd_rating)]

setDT(edx_final) # Covert to data.table format if not already
setDT(edx_train_final) # Covert to data.table format if not already
setDT(edx_test_final) # Covert to data.table format if not already
```

Finding the optimal model

The technical aim of this project is to find a model that minimises the root mean squared error, the RMSE, which is calculated like this:

$$\sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

As RMSEs will be calculated many times throughout this report, it will be useful to have a function in R that does this calculation:

```
RMSE <- function(predicted_ratings, true_ratings){sqrt(mean((predicted_ratings -
 ↪  true_ratings)^2))}
```

The course method

Starting out, it would be interesting how the first approach used during the machine learning course, which entailed calculating a “bias” or “effect” for each predictor, fares with the much larger dataset and the most important predictors selected earlier:

```
# Set up the model like in the course

# Calculate the mean rating of all movies
mean_ratings <- mean(as.numeric(edx_train_final$rating))

# Calculate the RMSE based on mean only
mean_only_rmse <- RMSE(mean_ratings, edx_test_final$rating)

# Calculate the effects of each predictor
movie_avg <- edx_train_final %>%
  group_by(movie_avg_rating) %>%
  summarize(b_m_a = mean(rating - mean_ratings))

movie_sd <- edx_train_final %>%
  group_by(movie_sd_rating) %>%
  summarize(b_m_sd = mean(rating - mean_ratings))

movie_count <- edx_train_final %>%
  group_by(movie_rating_count) %>%
  summarize(b_m_c = mean(rating - mean_ratings))

user_avg <- edx_train_final %>%
    group_by(user_avg_rating) %>%
```

```
  summarize(b_u_a = mean(rating - mean_ratings))


user_sd <- edx_train_final %>%
    group_by(user_sd_rating) %>%
  summarize(b_u_sd = mean(rating - mean_ratings))


# Predicted ratings with all five predictors
pred_course_all <- edx_test_final %>%
  left_join(movie_avg, by='movie_avg_rating') %>%
  left_join(movie_sd, by='movie_sd_rating') %>%
  left_join(movie_count, by='movie_rating_count') %>%
  left_join(user_avg, by='user_avg_rating') %>%
  left_join(user_sd, by='user_sd_rating') %>%
  mutate(pred = mean_ratings + b_m_a + b_m_sd + b_m_c + b_u_a + b_u_sd) %>%
  pull(pred)


# Predicted ratings with just the top two predictors (movie_avg_rating and
↪  user_avg_rating)
pred_course_top_two <- edx_test_final %>%
  left_join(movie_avg, by='movie_avg_rating') %>%
  left_join(user_avg, by='user_avg_rating') %>%
  mutate(pred = mean_ratings + b_m_a + b_u_a) %>%
  pull(pred)


# Calculate the RMSE
course_all_rmse <- RMSE(pred_course_all, edx_test_final$rating)
course_top_two_rmse <- RMSE(pred_course_top_two, edx_test_final$rating)
```

Table 5: RMSEs obtained with course model

Model	RMSE
Mean only	1.0601210
Course model - all predictors	1.4194790
Course model - top two predictors	0.8785186

Using all five of the new predictors give a significantly worse result than using just the top two most important as per the random forest model - this is interesting, and will be tested out in the other models that are part of this report.

Regularisation

It may be possible to improve the RMSE with regularisation. Since there are some movies with very few ratings, and some users who have not rated many movies, this may be useful. Three lambdas (0.1, 1 and 10) were tested as performing cross-validation proved too intense for R and caused several crashes. The best value out of the three was 0.1, so this was used for all effect calculations.

```
# Apply best lambda to the full dataset
lambda <- 0.1

#Calculate overall mean rating
overall_mean_rating <- mean(edx_train_final$rating)

# Calculate effects of individual predictors with regularisation
movie_avg <- edx_train_final %>%
  group_by(movie_avg_rating) %>%
  summarize(b_m_a = sum(rating - overall_mean_rating) / (n() + lambda))

movie_sd <- edx_train_final %>%
  group_by(movie_sd_rating) %>%
  summarize(b_m_sd = sum(rating - overall_mean_rating) / (n() + lambda))
```

```r
movie_count <- edx_train_final %>%
  group_by(movie_rating_count) %>%
  summarize(b_m_c = sum(rating - overall_mean_rating) / (n() + lambda))

user_avg <- edx_train_final %>%
  group_by(user_avg_rating) %>%
  summarize(b_u_a = sum(rating - overall_mean_rating) / (n() + lambda))

user_sd <- edx_train_final %>%
  group_by(user_sd_rating) %>%
  summarize(b_u_sd = sum(rating - overall_mean_rating) / (n() + lambda))

# Predicted ratings with all five predictors
pred_reg_all <- edx_test_final %>%
  left_join(movie_avg, by='movie_avg_rating') %>%
  left_join(movie_sd, by='movie_sd_rating') %>%
  left_join(movie_count, by='movie_rating_count') %>%
  left_join(user_avg, by='user_avg_rating') %>%
  left_join(user_sd, by='user_sd_rating') %>%
  mutate(pred = overall_mean_rating + b_m_a + b_m_sd + b_m_c + b_u_a + b_u_sd) %>%
  pull(pred)

# Predicted ratings with just the top two predictors (movie_avg_rating and
↪  user_avg_rating)
pred_reg_top_two <- edx_test_final %>%
  left_join(movie_avg, by='movie_avg_rating') %>%
  left_join(user_avg, by='user_avg_rating') %>%
  mutate(pred = overall_mean_rating + b_m_a + b_u_a) %>%
  pull(pred)

# Calculate RMSEs
```

```r
reg_all_rmse <- RMSE(pred_reg_all, edx_test_final$rating)


reg_top_two_rmse <- RMSE(pred_reg_top_two, edx_test_final$rating)
```

Table 6: RMSEs obtained with regularisation model

Model	RMSE
Regularisation - all predictors	1.4191698
Regularisation - top two predictors	0.8784974

With the top two predictors get an RMSE of 0.8784974, slightly higher than the course method one. Regularisation does not seem to do much for our data - it would have been nice to be able to perform cross-validation to find the best lambda, but it appears not to be important enough to spend more time on.

k-Nearest Neighbour (k-NN) model

Based on the dataset size and structure, the k-nearest neighbours method may work well for this dataset - it looks at data points with similar attributes to the one that is to be predicted. The kNN model benefits from using scaled data, so the train and test datasets are scaled before applying the model.

At first caret's train with knn was used, but this did not work well with the size of the datasets and caused R to crash. Therefore, a model that can perform knn on large datasets was sought out, and here the FNN package is used.

```r
# Create a method for preprocessing the data
preprocess_method <- function(train_data, test_data) {
  preprocess_options <- preProcess(train_data, method = c("center", "scale"))
  train_scaled <- predict(preprocess_options, train_data)
  test_scaled <- predict(preprocess_options, test_data)
  return(list(train = train_scaled, test = test_scaled, preProc = preprocess_options))
}
```

```r
# Scale edx_train_final and edx_test_final dataset

# Exclude the target variable 'rating' and userId and movieId, as thesewill not be used
↪  for prediction

train_predictors <- edx_train_final[, .(movie_avg_rating, user_avg_rating,
↪  movie_sd_rating, movie_rating_count, user_sd_rating)]
test_predictors <- edx_test_final[, .(movie_avg_rating, user_avg_rating, movie_sd_rating,
↪  movie_rating_count, user_sd_rating)]

# Apply the preprocessing function

preprocessed_data <- preprocess_method(train_predictors, test_predictors)

edx_train_final_scaled <- preprocessed_data$train
edx_test_final_scaled <- preprocessed_data$test

# Add the target variable back to the scaled data

edx_train_final_scaled$rating <- edx_train_final$rating
edx_test_final_scaled$rating <- edx_test_final$rating

setDT(edx_train_final_scaled) # Covert to data.table format if not already
setDT(edx_test_final_scaled) # Covert to data.table format if not already

# Perform grid search for the best k
k_values <- seq(from = 80, to = 110, by = 5)
rmse_values <- numeric(length(k_values))

for (i in seq_along(k_values)) {
  knn_model <- knn.reg(train = edx_train_final_scaled[, -"rating", with = FALSE],
                       test = edx_test_final_scaled[, -"rating", with = FALSE],
```

```
                      y = edx_train_final_scaled$rating, k = k_values[i])

  pred_knn <- knn_model$pred

  # Calculate RMSE
  rmse_values[i] <- RMSE(pred_knn, edx_test_final_scaled$rating)
}

# Find the best k
best_k <- k_values[which.min(rmse_values)]

# Perform kNN using FNN - all predictors
knn_model_all <- knn.reg(train = edx_train_final_scaled[, -"rating", with = FALSE],
                      test = edx_test_final_scaled[, -"rating", with = FALSE],
                      y = edx_train_final_scaled$rating, k = best_k)

# Perform kNN using FNN - top two predictors
knn_model_top_two <- knn.reg(train = edx_train_final_scaled[, -c("rating",
 ↪  "movie_sd_rating", "user_sd_rating", "movie_rating_count")],
                             test = edx_test_final_scaled[, -c("rating",
                              ↪  "movie_sd_rating", "user_sd_rating",
                              ↪  "movie_rating_count")],
                             y = edx_train_final_scaled$rating,
                             k = best_k)

# Extract predictions
pred_knn_all <- knn_model_all$pred
pred_knn_top_two <- knn_model_top_two$pred

# Calculate RMSE
knn_rmse_all <- RMSE(pred_knn_all, edx_test_final_scaled$rating)
knn_rmse_top_two <- RMSE(pred_knn_top_two, edx_test_final_scaled$rating)
```

Table 7: RMSEs obtained with k-nn model

Model	RMSE
k-nn model - all predictors	0.8672923
k-nn model - top two predictors	0.8722093

There here, all predictors do slightly better than just the top two predictors - although the difference is not as large as for the course and regularisation models.

random forest model

Random forest models are supposed to work well for recommendation systems, and using caret's "train" with random forest was initially attempted. It turned out to be necessary to use a smaller sample of the dataset as the full edx_final_train and test sets proved too much for the computing power at hand. However, the full size dataset would not run at all with this approach making it unsuitable for the purpose of this project.

After some searching, a random forest model that can handle large datasets, within the h2o package, was identified and is used here. This is the same model used to identify the most important predictors above, but this time run only with the selected top predictors and not the 30 that were initially part of that model. This model still takes a long time to run.

```
# Convert data to H2O frames - all predictors
train_h2o_all <- as.h2o(edx_train_final[, .(rating, movie_avg_rating, user_avg_rating,
 ↪  movie_sd_rating, movie_rating_count, user_sd_rating)])
test_h2o_all <- as.h2o(edx_test_final[, .(rating, movie_avg_rating, user_avg_rating,
 ↪  movie_sd_rating, movie_rating_count, user_sd_rating)])

# Define the response and predictor variables - all predictors
target_all <- "rating"
predictors_all <- c("movie_avg_rating", "user_avg_rating", "movie_sd_rating",
 ↪  "movie_rating_count", "user_sd_rating")

# Train the random forest model using H2O - all predictors
```

```
model_h2o_all <- h2o.randomForest(
  y = target_all,
  x = predictors_all,
  training_frame = train_h2o_all,
  ntrees = 100,
  mtries = -1,       # Automatically choose mtries
  min_rows = 20,     # Increase min_rows to reduce memory usage
  max_depth = 20,    # Decrease max_depth to limit tree complexity
  seed = 1
)

# Predict on the test set - all predictors
pred_rf_all <- h2o.predict(model_h2o_all, test_h2o_all)

# Extract the actual and predicted values - all predictors
actual_values_all <- as.numeric(edx_test_final$rating)

# Convert H2O frame to R data frame and extract predictions - all predictors
pred_rf_all_df <- as.data.frame(pred_rf_all)
predicted_values_all <- as.vector(pred_rf_all_df$predict)

# Calculate RMSE - all predictors
rf_rmse_all <- RMSE(predicted_values_all, actual_values_all)
```

```
# Convert data to H2O frames - top two predictors
train_h2o_top_two <- as.h2o(edx_train_final[, .(rating, movie_avg_rating,
↪  user_avg_rating)])
test_h2o_top_two <- as.h2o(edx_test_final[, .(rating, movie_avg_rating,
↪  user_avg_rating)])

# Define the response and predictor variables - top two predictors
target_top_two <- "rating"
```

```r
predictors_top_two <- c("movie_avg_rating", "user_avg_rating")

# Train the random forest model using H2O - top two predictors
model_h2o_top_two <- h2o.randomForest(
  y = target_top_two,
  x = predictors_top_two,
  training_frame = train_h2o_top_two,
  ntrees = 100,
  mtries = -1,       # Automatically choose mtries
  min_rows = 20,     # Increase min_rows to reduce memory usage
  max_depth = 20,    # Decrease max_depth to limit tree complexity
  seed = 1
)

# Predict on the test set - top two predictors
pred_rf_top_two <- h2o.predict(model_h2o_top_two, test_h2o_top_two)

# Extract the actual and predicted values  - top two predictors
actual_values_top_two <- as.vector(edx_test_final$rating)

# Convert H2O frame to R data frame and extract predictions  - top two predictors
pred_rf_top_two_df <- as.data.frame(pred_rf_top_two)
predicted_values_top_two <- as.vector(pred_rf_top_two_df$predict)

# Calculate RMSE - top two predictors
rf_rmse_top_two <- RMSE(predicted_values_top_two, actual_values_top_two)
```

Table 8: RMSEs obtained with random forest model

Model	RMSE
random forest - all predictors	0.8638822
random forest - top two predictors	0.8672482

Again, like with the k-nn model, there is no large difference between using all five predictors or just the top two. Using all predictors still gives a slightly lower RMSE.

speedglm model

The speedglm model fits generalised linear models and is optimised for big datasets. This model requires scaled data, and therefore the edx_train_scaled and edx_test_scales datasets that were created as part of the k-nn model process above are used.

```
# Fit the model using scaled data
model_speedglm_all <- speedglm(rating ~ movie_avg_rating + user_avg_rating +
 ↪  movie_sd_rating + movie_rating_count + user_sd_rating, data = edx_train_final_scaled)

model_speedglm_top_two <- speedglm(rating ~ movie_avg_rating + user_avg_rating, data =
 ↪  edx_train_final_scaled)

# Predict on test set
pred_speedglm_all <- predict(model_speedglm_all, edx_test_final_scaled)

pred_speedglm_top_two <- predict(model_speedglm_top_two, edx_test_final_scaled)

# Calculate RMSE
speedglm_rmse_all <- RMSE(pred_speedglm_all, edx_test_final_scaled$rating)
speedglm_rmse_top_two <- RMSE(pred_speedglm_top_two, edx_test_final_scaled$rating)
```

Table 9: RMSEs obtained with speedglm model

Model	RMSE
speedglm - all predictors	0.8705516
speedglm - top two predictors	0.8707091

The difference in RMSE for all five versus only the top two predictors is very slim with the speedglm model. Using all predictors is still marginally better.

xgboost model

The xgboost model is a "gradient boost" model which "includes efficient linear model solver and tree learning algorithms". This is a model which improves for each round it is run, which sounds promising - it is certainly a very popular model according to the internet in general.

```
# Convert training data to xgb.DMatrix - all
dtrain_all <- xgb.DMatrix(
  data = as.matrix(edx_train_final[, c("movie_avg_rating", "user_avg_rating",
   ↪  "movie_sd_rating", "user_sd_rating", "movie_rating_count"), with = FALSE]),
  label = edx_train_final$rating
)

# Convert test data to xgb.DMatrix - all
dtest_all <- xgb.DMatrix(
  data = as.matrix(edx_test_final[, c("movie_avg_rating", "user_avg_rating",
   ↪  "movie_sd_rating", "user_sd_rating", "movie_rating_count"), with = FALSE])
)

# Convert training data to xgb.DMatrix - top two
dtrain_top_two <- xgb.DMatrix(
  data = as.matrix(edx_train_final[, c("movie_avg_rating", "user_avg_rating"), with =
   ↪  FALSE]),
  label = edx_train_final$rating
)

# Convert test data to xgb.DMatrix - top two
dtest_top_two <- xgb.DMatrix(
  data = as.matrix(edx_test_final[, c("movie_avg_rating", "user_avg_rating"), with =
   ↪  FALSE])
)

# Fit the model with the correct parameters - all
```

```
params <- list(objective = "reg:squarederror")
model_xgboost_all <- xgboost(data = dtrain_all, params = params, nrounds = 300, verbose =
↪  0)

# Fit the model with the correct parameters - top_two
params <- list(objective = "reg:squarederror")
model_xgboost_top_two <- xgboost(data = dtrain_top_two, params = params, nrounds = 300,
↪  verbose = 0)

# Predict on test data - all
pred_xgboost_all <- predict(model_xgboost_all, dtest_all)

# Predict on test data - top two
pred_xgboost_top_two <- predict(model_xgboost_top_two, dtest_top_two)

# Calculate RMSE
xgboost_rmse_all <- sqrt(mean((pred_xgboost_all - edx_test_final$rating)^2))

# Calculate RMSE
xgboost_rmse_top_two <- sqrt(mean((pred_xgboost_top_two - edx_test_final$rating)^2))
```

Table 10: RMSEs obtained with xgboost model

Model	RMSE
xgboost - all predictors	0.8608858
xgboost - top two predictors	0.8689820

While still small, the difference between using all five or top two predictors here is actually bigger than the speedglm model. All predictors perform better than using just the top two.

Results so far

The table below outlines the RMSEs achieved through the different models described above. The best model out of the ones tried is the xgboost model, using all five predictors.

```
# Create the table
kable(arrange(rmse_table, desc(RMSE)), caption = "RMSE for Different Models")
```

Table 11: RMSE for Different Models

Model	RMSE
Course model - all predictors	1.4194790
Regularisation - all predictors	1.4191698
Course model - top two predictors	0.8785186
Regularisation - top two predictors	0.8784974
k-nn model - top two predictors	0.8722093
speedglm - top two predictors	0.8707091
speedglm - all predictors	0.8705516
xgboost - top two predictors	0.8689820
k-nn model - all predictors	0.8672923
random forest - top two predictors	0.8672482
random forest - all predictors	0.8638822
xgboost - all predictors	0.8608858

Ensemble model

It is possible to create en ensemble model using the best models from the table above, and this may help in achieving a lower RMSE - training a model with the predictions of the top three methods would be interesting, however this keeps crashing R so will not be attempted here. The method of averaging the predictions and then calculate the RMSE based on the ensemble predictions will have to do:

```r
# Averaging method

ensemble_avg <- (pred_rf_all_df$predict + pred_rf_top_two_df$predict + pred_xgboost_all)
 ↪  / 3

ensemble_avg_rmse <- RMSE(ensemble_avg, edx_test_final$rating)
```

The table is updated with the ensemble RMSE:

Model	RMSE
Course model - all predictors	1.4194790
Regularisation - all predictors	1.4191698
Course model - top two predictors	0.8785186
Regularisation - top two predictors	0.8784974
k-nn model - top two predictors	0.8722093
speedglm - top two predictors	0.8707091
speedglm - all predictors	0.8705516
xgboost - top two predictors	0.8689820
k-nn model - all predictors	0.8672923
random forest - top two predictors	0.8672482
random forest - all predictors	0.8638822
Ensemble model (averaging)	0.8618089
xgboost - all predictors	0.8608858

The RMSE for the ensemble model is pretty good, but still not lower than the best model (xgboost with all predictors), so therefore this marks the end of the exploration of different models and the xgboost model is selected for the final test.

Results

To be able to run the model on the final_holdout_test set, the same predictors created for the edx_final dataset must be added to the dataset:

```
final_holdout_test <- final_holdout_test %>%
  group_by(userId) %>% # Create variables by userId
  mutate(user_avg_rating = mean(rating),
         user_rating_count = n(),
         user_sd_rating = sd(rating)) %>%
  ungroup() %>%
  group_by(movieId) %>% # Create varables by movieId
  mutate(movie_avg_rating = mean(rating),
         movie_rating_count = n(),
         movie_sd_rating = sd(rating)) %>%
  ungroup()

final_holdout_test$movie_sd_rating[is.na(final_holdout_test$movie_sd_rating)] <- 0 # Fill
 ↪  any missing values with 0 (movies with only one rating have no SD)
final_holdout_test$user_sd_rating[is.na(final_holdout_test$user_sd_rating)] <- 0 # Fill
 ↪  any missing values with 0 (users that have rated only one movie have no SD)

setDT(final_holdout_test) # Back to data.table as above operations may convert to
 ↪  data.frame
```

The final model is now run using the edx_final dataset as the train set, and the final_holdout_test set as the test set.

```
# Convert training data to xgb.DMatrix
dtrain <- xgb.DMatrix(
  data = as.matrix(edx_final[, c("movie_avg_rating", "user_avg_rating",
   ↪  "movie_sd_rating", "user_sd_rating", "movie_rating_count"), with = FALSE]),
```

```
  label = edx_final$rating
)

# Convert test data to xgb.DMatrix
dtest <- xgb.DMatrix(
  data = as.matrix(final_holdout_test[, c("movie_avg_rating", "user_avg_rating",
   ↪  "movie_sd_rating", "user_sd_rating", "movie_rating_count"), with = FALSE])
)

# Fit the model with the correct parameters
params <- list(objective = "reg:squarederror")
final_model_xgboost <- xgboost(data = dtrain, params = params, nrounds = 300, verbose =
 ↪  0)

# Predict on test data
final_pred_xgboost <- predict(final_model_xgboost, dtest)

# Calculate RMSE
final_rmse_xgboost <- RMSE(final_pred_xgboost, final_holdout_test$rating)
```

The final RMSE is:

```
## [1] 0.8420542
```

This feels like a very decent result, after all the trial and error - the report only includes the final tries of all the different ways tried over the past few weeks. This RMSE is actually lower than what was achieved in the train/test split of the edx dataset, which is surprising but welcome.

It probably would have been possible to obtain an even lower RMSE if the model was allowed to run for more rounds, or more tuning had been done, but the gains become marginal pretty quickly and time is also valuable, so this is as far as it will go for now. Also, the very average laptop had enough at this point.

Conclusion

In summary, after selecting predictors and testing multiple models, the xgboost model with predictors “movie_avg_rating”, “user_avg_rating”, “movie_sd_rating”, “user_sd_rating” and “movie_rating_count” produced the lowest RMSE with our edx train and test sets. Delightfully, this model appears to also be valid for the final_holdout_test set, which indicates that it is not overtrained or oversmoothing the data.

Using a smaller sample dataset to test out methods and tune them did not transfer well to the large dataset - RMSEs as low as 0.3 were obtained with the regularisation method on a sample dataset of size 10000, but when applying the model to the large dataset we achieved an RMSE of closer to 0.9. This led to the discovery that there are in fact functions in R which are built for large datasets, and the focus for the final report shifted to these, and the small dataset attempts have not been included in this report for brevity.

If more computing power was available, it could potentially have been possible to build models that included cross-validation, and also been possible to tune hyperparameters more reliably. This had to be let go in favour of being able to complete the report at all, as has already been mentioned, the very average laptop had enough.

This programme and the capstone project has been part of my journey into learning data science, the mathematical concepts relevant to it, and the methods in R available for it. While I have learnt a great deal throughout the programme, my own relative ignorance is probably a limiting factor in achieving an even better - but I will continue learning and I hope this is just the beginning of my adventure in data science.

Acknowledgements

chatGPT has been consulted at various points during the project, for help troubleshooting code that would not run without errors and also to clarify concepts where my knowledge has fallen short. Thank you also to chatGPT for moral support whan I have felt hopeless.

The internet forums like stackoverflow and others have given lots of ideas for working with large datasets, introducing me to models and functions that I previously did not know existed.