MovieLens Project

Anna Win-Mason

2025-04-29

Introduction

Recommendation systems provide predictions based on prior ratings that users have given to items using a machine learning approach.(Irizarry 2019) More specifically, a movie recommendation system is based on a set of user inputted data, which can throw up movie recommendations based on past movie preferences.

There can be two different high-level strategies used for movie recommendation systems.(Motefaker 2024) 1. Content-based filtering: where the past movie ratings of a user are used to predict similar movies for that user. 2. Collaborative filtering: where multiple users provide movie ratings. Movie predictions can be provided by filtering on users that have similar preferences, or by filtering on similar movies that users have rated.

Movie recommendation systems are used by companies like Netflix to deliver up and filter content. There is a lot of money involved for companies like Netflix in attracting users to their streaming service and keeping them coming back with relevant recommendations.

In this project I will develop a movie recommendation system. The Netflix data is not available to the public, however the GroupLens research lab has generated a database with over 20 million ratings for over 27,000 movies by more than 138,000 users (GroupLens 2025b). A stable benchmark subset of this database with 10 million ratings, MovieLens 10M(GroupLens 2025a), will be used to make the computations easier.

An overview of the key steps used in the project:

1. Download and wrangle the MovieLens 10M dataset, creating a final hold-out test set to assess the accuracy of the final model.
2. Exploratory data analysis to understand the variables and their distributions in the dataset.
3. Create a test set and training set to develop and validate the recommendation system models.
4. Develop a recommendation system model by starting with a baseline algorithm, then add parameters in an iterative way to refine the algorithm towards a final model.
5. Use the root mean square error (RMSE) as a benchmark metric to assess and compare the effectiveness of each algorithm.
6. Make movie predictions with the final recommendation model.

Methods and analysis

The MovieLens 10M dataset

We start by downloading and preparing the MovieLens 10M dataset for use in developing the machine learning algorithm with the below code. The final model will be built with the *edx* dataset, and will be tested for prediction accuracy on the *final_holdout_test set*.

```r
# Create edx and final_holdout_test sets from the MovieLens 10M dataset

# Note: this process could take a couple of minutes

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

options(timeout = 120)

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

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

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

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

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

movielens <- left_join(ratings, movies, by = "movieId")

# Final hold-out test set will be 10% of MovieLens data

set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier

test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

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

Exploratory data analysis of *edx* dataset

Let's look into the *edx* dataset in more detail. Each row of data in the *edx* dataset corresponds to one ranking by one user for a movie.

```r
# the number of rows and columns in edx
dim(edx)
```

```
## [1] 9000055       6
```

There are six variables for each row of data in *edx* detailed in table 1. Whereas table 2 shows the top four rows of the *edx* dataset.

Table 1: Definitions of *edx* variables

Variables	Definitions
userID	integer, ID for each unique user in dataset
movieId	integer, ID for each unique movie in dataset
rating	number, user movie rating from 0 - 5, including half star ratings
timestamp	integer, the time and date when a movie rating is added by a user
title	character, the title of the movie being rated
genres	character, one or more genres that the movie fits in

```r
# head of edx
head(edx, n = 4) %>% kable(caption = "Edx dataset")
```

Table 2: Edx dataset

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi

Data content of *edx* as motivation for a recommendation system

Let's look into the data itself. Table 3 shows the number of unique movies, users, and genre combinations. As you can see the data in *edx* account for ratings of just over 10K individual movies provided by about 70K individual users. If we were to multiply these two numbers we'd have about 750M ratings, but there are only ~ 9M rows of data in the dataset. Therefore, not every user has rated every movie, and there are gaps or NAs in the *edx* dataset. We can use all of the ratings data in *edx* as predictors for building a machine learning algorithm for a movie recommendation system.

```r
# The number of different movies
unique_movies <- edx %>% distinct(movieId)

# The number of different users
unique_users <- edx %>% distinct(userId)

# The number of different genre combinations
unique_genres <- edx %>% distinct(genres)
```

Table 3: Unique data variables in edx

unique_var	count
movies	10677
users	69878
genre combo's	797

Data distributions

We will now investigate the distribution of ratings by movies, users and genre in the *edx* dataset.

Ratings distributions of unique movies

We find that some movies have been rated many more times than others, as visualised in figure 1. This is not surprising given some movies will be blockbuster hits viewed by many people, and some will be bespoke art-house movies viewed by less people, and therefore receiving fewer ratings.

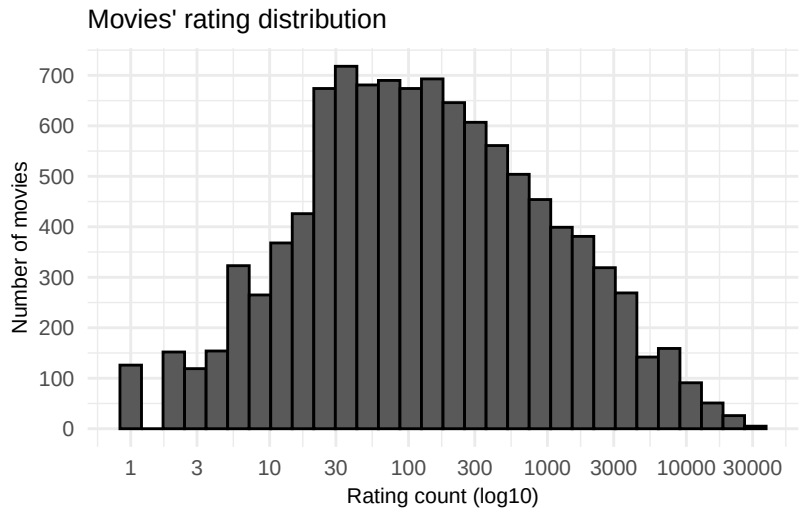

Figure 1: The distribution of the number of ratings per movie in edx

Ratings distributions of unique users

We also find that some users have been much more active providing ratings than others, illustrated by the following histogram in figure 2.

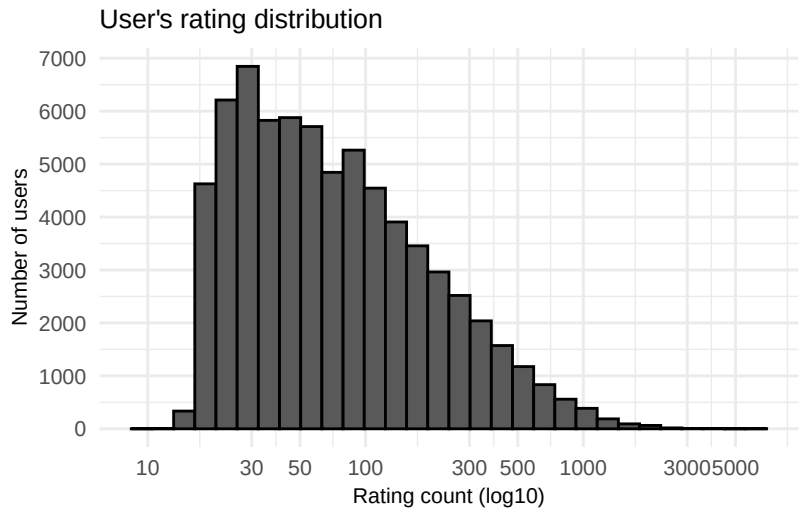

Figure 2: The distribution of the number of ratings per user in edx

Individual genres

Next we will look at movie genres. In the *edx* dataset most movies are classified under multiple genres. So to determine how many of each genre are present in the dataset we can use this R code:

```
# detect how many of each genre in the dataset using the sapply() and str_detect()
# list of most common genres
genres = c("Action", "Adventure", "Animation",
           "Children", "Comedy", "Crime",
           "Documentary", "Drama", "Fantasy",
           "Film-Noir", "Horror", "Musical",
           "Mystery", "Romance", "Sci-Fi",
           "Thriller", "War", "Western")
# create tibble of genres and count, where genres = each individual genre and
# count = sapply function that detects and sums each genre string contained in the
# genres column of edx
genres_count <- tibble(
  genres = genres,
  count = sapply(genres, function(g) {
  sum(str_detect(edx$genres, g))
})
)
```

A plot of the number of individual movie genres in *edx* shows that there is wide variability (figure 3). The movie genres that feature the most in *edx* are *drama* and *comedy*, whereas the movie genres that feature the least are *documentary* and *film-noir*.

Ratings by year

We can also look at the number of ratings inputted by users by year. As can be seen in Figure 4, some years there were many more ratings than in other years, with peak years being 1996, 2000 and 2005.

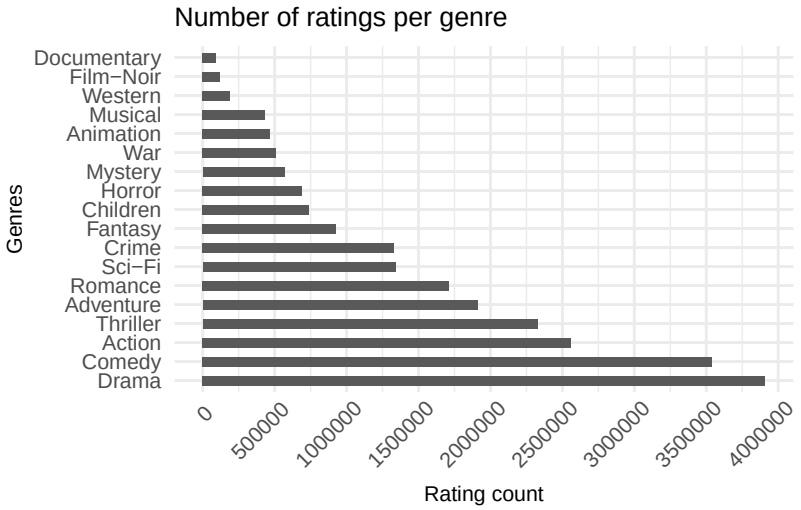

Figure 3: The number of individual genres in movie ratings in edx

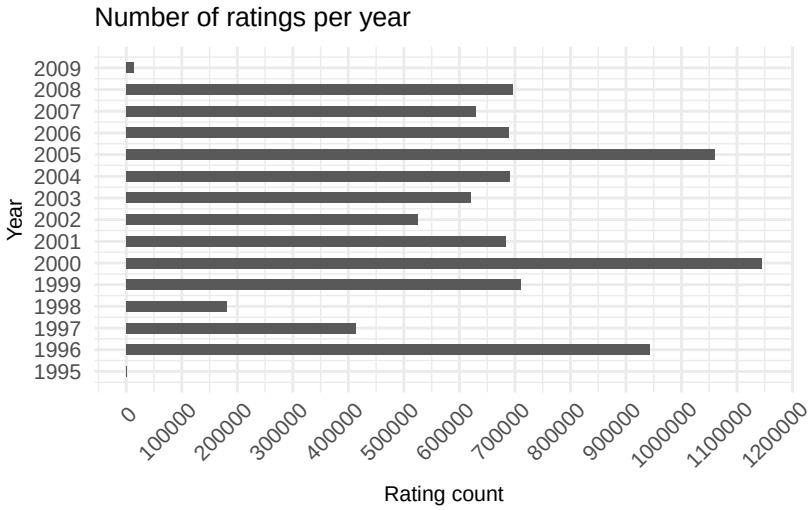

Figure 4: The number of movie ratings submitted by users per year

Machine learning strategy

Data as predictors

We've explored the inherent variety of the number of ratings by movie, user, genre and year in *edx*. For the movie recommendation system we will exploit the movie, user, genre variable effects or biases in an iterative fashion to build the final model (table 4). Although there is a variation in the number of ratings by year, to limit the scope of this project I've decided that the year variable won't be used in building the model.

Table 4: Characterstics to build the model

iteration	method_charactersistics
1	Average baseline
2	Movie effects
3	Movie + user effects
4	Movie + user + genre effects
5	Regularised movie effects
6	Regularised movie + user effects
7	Regularised movie + user + genre effects

Determing model accuracy

We need a metric to determine the accuracy of the predictions and to evaluate usefulness of the machine learning algorithms. One on the most common evaluation metrics is the root mean square error or RMSE.(Jonatasv 2024) RMSE measures the average difference between a model's actual and predicted values. Mathematically, it is the standard deviation of the differences. The lower the RMSE the less error in the model.

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

RMSE can be defined by the following R code:

```
# RMSE written out as an R function.
# RMSE is the sqrt of the mean of (true_ratings - predicted_ratings) squared
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Model validation

To develop the movie recommendation algorithm using machine learning methods, it is best practice to take the *edx* dataset and split it further into a *train_set* (80%) and a *test_set* (20%). This is needed to avoid over-training the data. For each method iteration we will use the train_set to develop the algorithm, then use the test_set to make ratings predictions using the algorithm, with the prediction accuracy evaluated using RMSE.

```
# The following code creates a train set and a test set from
# the edx MovieLens dataset.

#set the seed with rounding
```

```r
set.seed(1, sample.kind="Rounding")

# create the partition index using caret::createDataPartion function
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)

# use the partition index to create the train_set and test_set
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

# Use semi-join() function so that the test set doesn't include users
# and movies that are not also in the train set.
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

Building a linear model

The average baseline

We will first use a linear approach to model recommendation system using the following formula:

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

First we will define the baseline model to compare all subsequent models to.(Irizarry 2019) This first model assumes the same rating for all movies and for all users, with the true rating represented by μ, and all the differences explained by random variation with $\epsilon_{u,i}$.

We will determine the estimate of μ, which is $\hat{\mu}$, by taking the average of all the ratings in the train_set. Then determine the RMSE of this baseline average model with the test_set, using $\hat{\mu}$ as the parameter.

```r
# determine mu_hat as the average of all ratings in the train_set
mu_hat <- mean(train_set$rating)


# determine RSME of the baseline average model using the caret::RSME function on
# the test_set and with mu_hat as a parameter.
rmse_average <- RMSE(test_set$rating, mu_hat)
rmse_average
```

```
## [1] 1.059904
```

Plus movie effects

We can look at the distribution of the average movie ratings in the *train_set* as shown in figure 5. We see that some movies are rated higher than others, therefore an improvement to the baseline model can take this movie effect into account.

Let's add a term to the linear model b_i to represent the average ranking for movie i . As this extra term b_i contributes to reducing the error term $\epsilon_{u,i}$ it should improve the model performance.

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

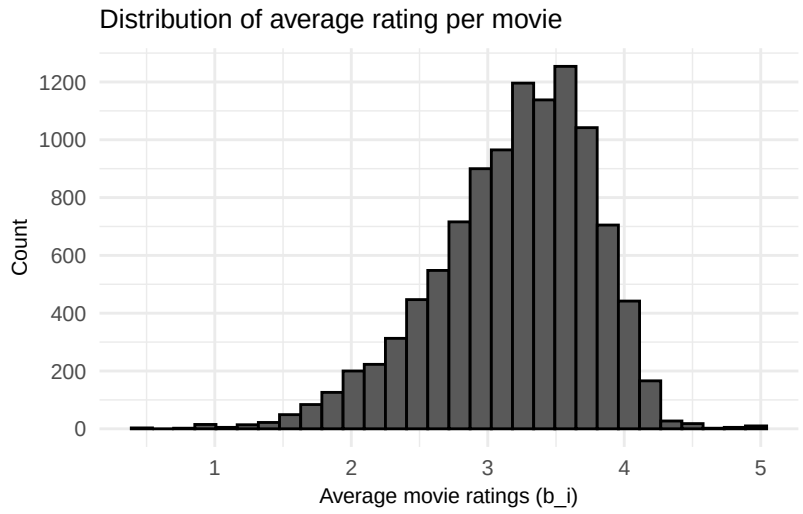

Figure 5: The distribution of average movie ratings in the train_set

To determine the estimate of b_i, or $\hat{b_i}$, we can use the least squares estimate. In this way we will compute the average of the *rating* minus the estimate of μ for each movie i, i.e. $Y_{u,i} - \hat{\mu}$.

```
# using least squares to estimate b_i_hat on the train_set
# to determine the average for each movie we use the group_by() function
movie_averages <- train_set %>%
  group_by(movieId) %>%
  summarise(b_i_hat = mean(rating - mu_hat))
```

The distribution of the estimates of b_i vary a lot with a range from -3 to 1.5 (figure 6).

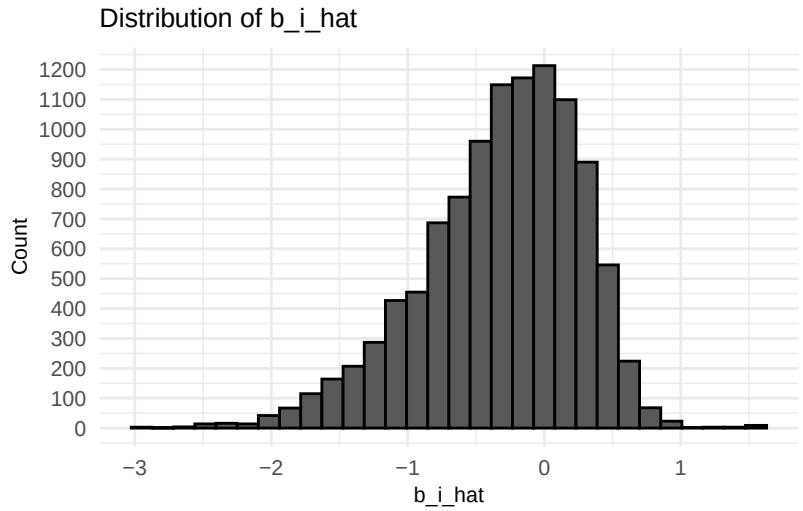

Figure 6: The distributution of the estimation of b_i in the train_set

We can now use the computed movie averages to build on the baseline average model and determine the predicted ratings of the movie effects model. Then we measure the RMSE of the movie effects model to evaluate whether adding the parameter $\hat{b_i}$ has improved the prediction of the model.

```r
# predicting the ratings using the test_set by joining the baseline model
# using left_join() to the movie averages, and pulling b_i_hat for evaluation
pred_ratings_b_i <- mu_hat + test_set %>%
  left_join(movie_averages, by = "movieId") %>%
  pull(b_i_hat)

# evaluating the plus movie effects model predictions using caret::RMSE
# on the test_set rating
rmse_movie_effects <- RMSE(pred_ratings_b_i, test_set$rating)
rmse_movie_effects
```

```
## [1] 0.9437429
```

We see that there has been some improvement to the model prediction as the RMSE is less than the baseline average. There are more improvements to make though.

Plus user effects

Let's look at the distribution of average user ratings in the dataset (figure 7). We can see there is variability in the ratings with some users giving high movie ratings and some really not liking what they've watched. Therefore, we can use this user variability to further improve the model.

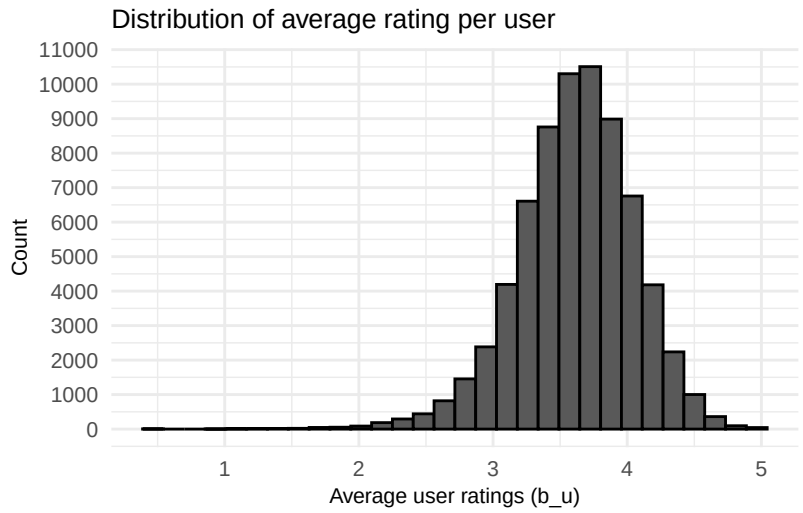

Figure 7: The distribution of average user ratings in the train_set

We will look at adding in the user effects b_u to further reduce the error term $\epsilon_{u,i}$ and improve the linear model:

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

To determine the estimate of b_u, or $\hat{b_u}$, we can use the least squares estimate. In this way we will compute the average of the *rating* minus the estimate of b_i and μ for each user u, i.e. $Y_{u,i} - \hat{b_i} - \hat{\mu}$.

```r
# using least squares to estimate b_u_hat on the train_set
user_averages <- train_set %>%
# left_join with the movie_averages dataset to add in the b_i_hat field
  left_join(movie_averages, by = "movieId") %>%
# to determine the average for each user we use the group_by() function
  group_by(userId) %>%
# summarise b_u_hat as the mean of rating - mu_hat - b_i_hat
  summarise(b_u_hat = mean(rating - mu_hat - b_i_hat))
```

The distribution of the estimates of b_u is centered across 0, and has a bit less variability than $\hat{bi}$ as can be seen in figure 8.

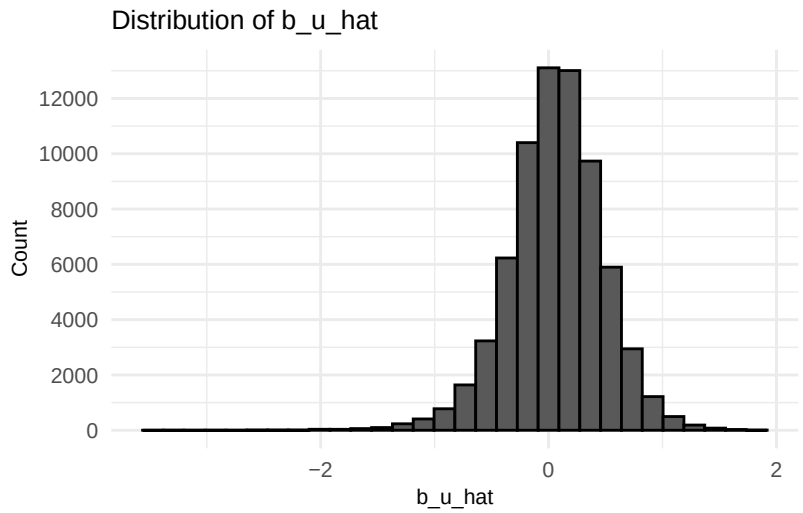

Figure 8: The distributution of the estimation of b_u in the train_set

We can now use the computed user averages to build on the movie effects model and determine the predicted ratings. Then we will measure the RMSE of this third iteration of the model to evaluate whether adding in the user effects with the parameter $\hat{b_u}$ has improved the model prediction.

```r
# predicting the ratings using the test_set, by joining the movie effects model
# using left_join() to the movie averages, and to join to the user_averages
# predicted ratings is the average + movie_average + user_average
predicted_ratings_b_u_hat <- test_set %>%
  left_join(movie_averages, by  = "movieId") %>%
  left_join(user_averages, by = "userId") %>%
  mutate(predicted = mu_hat + b_i_hat + b_u_hat) %>%
  pull(predicted)

# evaluating the plus user effects model predictions using caret::RMSE
rmse_user_effects<- RMSE(predicted_ratings_b_u_hat, test_set$rating)
# print rmse_user_effects
rmse_user_effects
```

```
## [1] 0.865932
```

There has been further improvement to the prediction accuracy by adding in user effects to the linear model.

Plus genre effects

The genres of each movie are another variable we can use as a parameter, b_g, to further improve the linear model and reduce the error term $\epsilon_{u,i}$:

$$Y_{u,i} = \mu + b_i + b_u + b_g + \epsilon_{u,i}$$

Here for simplification we will be using the genre combinations for each movie, rather than each detecting individual genre. To determine the estimate of b_g, or $\hat{b_g}$, once again we use the least squares estimate. We will compute the average of the *rating* minus the estimates of b_i, b_u and μ for each genre combination g, i.e. $Y_{u,i} - \hat{b_i} - \hat{b_u} - \hat{\mu}$.

```
# using least squares to estimate b_g_hat on the train_set
genre_averages <- train_set %>%
# left_join with the movie_averages dataset to add in the b_i_hat field
  left_join(movie_averages, by ="movieId") %>%
# left_join with the movie_averages dataset to add in the b_u_hat field
  left_join(user_averages, by = "userId") %>%
# group by genres
  group_by(genres) %>%
# summarise b_g_hat as the mean of rating - mu_hat - b_i_hat - b_u_hat
  summarise(b_g_hat = mean(rating - mu_hat - b_i_hat - b_u_hat))
```

The distribution of the estimates of b_g is less varied than that of $\hat{b_i}$ and $\hat{bu}$ (figure 9). However, we should still see some improvement to the model prediction by adding in genre effects.

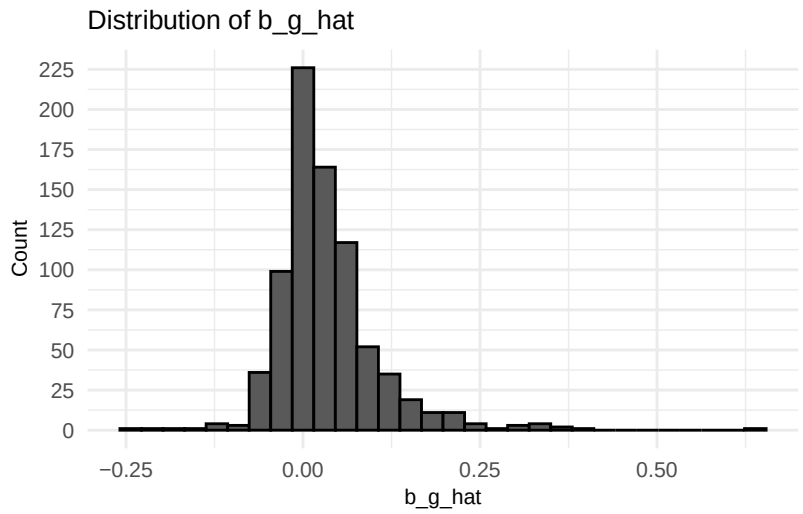

Figure 9: The distributution of the estimation of b_g in the train_set

We can now use the genre averages, which was computed on the train_set building on the movie + user effects model, to make predicted ratings using the test_set. Then we will measure the RMSE of this fourth iteration of the model to evaluate whether adding in the genre effects with the parameter $\hat{b_g}$ has improved the model prediction.

```
# predicting the ratings using the test_set
predicted_ratings_b_g_hat <- test_set %>%
# using left_join() to the movie averages
```

```
  left_join(movie_averages, by  = "movieId") %>%
# using left_join() to the user averages
  left_join(user_averages, by = "userId") %>%
# using left_join() to the genre averages
  left_join(genre_averages, by = "genres") %>%
# predicted ratings is the estimated average + movie_average + user_average
#  + genre_average
  mutate(predicted = mu_hat + b_i_hat + b_u_hat + b_g_hat) %>%
  pull(predicted)

# evaluating the plus genre effects model predictions using caret::RMSE
rmse_genre_effects <- RMSE(predicted_ratings_b_g_hat, test_set$rating)
# print rmse_genre_effects
rmse_genre_effects
```

```
## [1] 0.8655941
```

The RMSEs of the four linear model iterations so far are depicted in table 5. The accuracy of the model predictions have improved from the first average baseline to the fourth movie + user + genre effects linear model, with the RMSE dropping from 1.059904 to 0.865594, respectively. However, we can make further improvements to model prediction accuracy.

Table 5: RMSE improvements over four models

iteration	method	RMSE
1	Average baseline	1.0599043
2	Movie effects	0.9437429
3	Movie + user effects	0.8659320
4	Movie + user + genre effects	0.8655941

Regularisation of movie effects

To further improve the model, we can use regularisation. This approach allows for penalising the large estimates that come from small sample sizes, i.e. when a movie has only a few ratings, or when a user has only submitted a few ratings etc.(Irizarry 2019)

To investigate why regularisation can help we will look at the top ten best recommended movies based on the movie effects model.

```
# assign the unique movie_titles in the edx dataset
movie_titles <- edx %>%
# select movieID and title
  select(movieId, title) %>%
# distinct
  distinct()

# Ten best movies in movie_averages
best_ten_i <- movie_averages %>%
# left_join movie_averages to movie_titles
  left_join(movie_titles, by="movieId") %>%
# arrange the table by greatest b_i_hat
  arrange(desc(b_i_hat)) %>%
```

```
# select title and b_i_hat
  select(title, b_i_hat) %>%
# select the top 10 rows
  slice(1:10) %>%
# pull the title
  pull(title)
```

Table 6: Ten best movies from movie effects model

best_ten	movie_titles
1	Hellhounds on My Trail (1999)
2	Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)
3	Satan's Tango (Sátántangó) (1994)
4	Shadows of Forgotten Ancestors (1964)
5	Money (Argent, L') (1983)
6	Fighting Elegy (Kenka erejii) (1966)
7	Sun Alley (Sonnenallee) (1999)
8	Aerial, The (La Antena) (2007)
9	Blue Light, The (Das Blaue Licht) (1932)
10	More (1998)

We can see that the top ten movies are all relatively obscure (table 6). If we look at how many times these movies have been rated this will shed some light on why this is the case.

```
# number of ratings for the ten best recommended movies
number_ratings <- train_set %>%
# count movieID ratings in train set
  count(movieId) %>%
# left join movie averages to get b_i_hat
  left_join(movie_averages) %>%
# left join movie titles for distinct movies
  left_join(movie_titles, by="movieId") %>%
# arrange the table by greatest b_i_hat
  arrange(desc(b_i_hat)) %>%
# select the top 10 rows
  slice(1:10) %>%
# pull the number of rows
  pull(n)
```

Table 7: Number of ratings for best ten movies

best_ten	number_of_movie_ratings
1	1
2	3
3	2
4	1
5	1
6	1
7	1
8	1

best_ten	number_of_movie_ratings
9	1
10	6

We can see in table 7 that there is a low number of ratings for these top ten movies, sometimes just one reviewer. These small sample sizes of ratings can lead to high uncertainty in the estimates b_i.

Regularisation is a tool that allows the large estimates that come from small sample sizes to be penalised using a formula that uses the least squares approach, but which introduces a penalty term. This penalty term includes a tuning parameter, lambda λ.

$$\hat{b_i}(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

To estimate b_i we need to do full cross-validation to determine the optimum value of λ. We also need to incorporate n_i the number of ratings per movie i. The R code for this cross validation is below:

```
# cross validation to tune for which value of lambda to use to regularise the
# movie effects model

# define lambdas
lambdas <- seq(0, 10, 0.1)

# this function uses sapply and returns the RMSE of the regularised
# movie effects model for each value of lambda in lambdas
tune_lambda <- sapply(lambdas, function(lambda){

  # compute mu_hat as the mean of rating in the train_set
  mu_hat <- mean(train_set$rating)

  # compute b_i_hat with train_set, group_by movieID
  # calculation using each value of lambda in lambdas
  reg_movie_effects <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i_hat = sum(rating - mu_hat)/(n()+lambda))

  # predict ratings against the test set
  # using calculated b_i_hat and mu_hat
  predicted_ratings <- test_set %>%
    left_join(reg_movie_effects, by = "movieId") %>%
    mutate(prediction = mu_hat + b_i_hat) %>%
    pull(prediction)

# find nd return RMSE using caret::RMSE with predicted ratings and the test_set rating
return(RMSE(predicted_ratings, test_set$rating))

})
```

We can visualise the value of lambda that provides the lowest RMSE for the regularised movie effects model in figure 10.

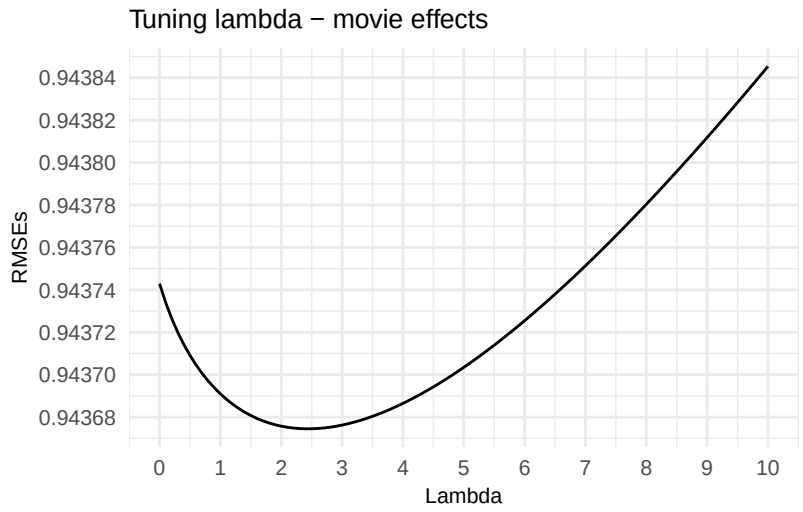

Figure 10: Tuning lambda to minimise the RMSE for regularised movie effects

```
# the best lambda is the one that minimises RMSE from function tune_lambda
best_lambda <- lambdas[which.min(tune_lambda)]
# print the best lambda
best_lambda
```

```
## [1] 2.4
```

```
# the RMSE of regularised movie effects is the minimal RMSE from tune_lambda
regularised_rmse_movie_effects <- min(tune_lambda)
# print the RMSE
regularised_rmse_movie_effects
```

```
## [1] 0.9436745
```

Note that the RMSE of the regularised movie effects algorithm is very similar to just the movie effects (table 5, iteration 2: 0.9437429). Though we can check if this updated regularised movie effects model has improved the recommendation of movies by looking at the top ten best movie predictions. First we run the algorithm on the train_set:

```
# using the best lambda
reg_just_movie_effects <- train_set %>%
  group_by(movieId) %>%
  summarise(b_i_hat = sum(rating - mu_hat)/(n()+best_lambda))
```

Then use the regularised movie effects algorithm to pull the top ten recommended movies. Table 8 shows the ten best rated movies after regularisation. This top ten list is looking more realistic. There are less obscure movies, and the number of ratings per movie in table 9 has greatly increased for the most part. However, I'm sure we can improve the movie recommendation model further.

```
# best 10 movies with regularisation of movie effects
best_ten_movies <- reg_just_movie_effects %>%
```

```
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i_hat)) %>%
  select(title) %>%
  head(10) %>%
  pull(title)
```

Table 8: Ten best movies with regularised movie effects

best_ten	movie_titles
1	More (1998)
2	Shawshank Redemption, The (1994)
3	Godfather, The (1972)
4	Usual Suspects, The (1995)
5	Schindler's List (1993)
6	Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)
7	Rear Window (1954)
8	Casablanca (1942)
9	Double Indemnity (1944)
10	Third Man, The (1949)

```
# number of ratings for the ten best recommended movies
number_ratings_reg <- train_set %>%
# count movieID ratings in train set
  count(movieId) %>%
# left join reg_just_movie_effects to get b_i_hat_(lambda)
  left_join(reg_just_movie_effects) %>%
# left join movie titles for distinct movies
  left_join(movie_titles, by="movieId") %>%
# arrange the table by greatest b_i_hat
  arrange(desc(b_i_hat)) %>%
# select the top 10 rows
  slice(1:10) %>%
# pull the number of rows
  pull(n)
```

Table 9: Number of ratings for top ten movies using regularisation

best_ten	number_of_movie_ratings
1	6
2	22363
3	14107
4	17315
5	18567
6	3
7	6325
8	9027
9	1711
10	2420

Regularisation of movie + user effects

To improve the model further, we'll use regularisation of the algorithm incorporating movie plus user effects. We can use the formula below to find the estimation of regularised user effects.

$$\hat{b_u}\left(\lambda\right) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} \left(Y_{u,i} - \hat{\mu} - \hat{b_i}\right)$$

We will need to use cross validation again to tune the model for the best lambda as we've added in the parameter $\hat{b_u}\left(\lambda\right)$ to the algorithm.

```
# cross validation to tune for which value of lambda to use to regularise the
# movie + user effects model

# define lambdas
lambdas <- seq(0, 10, 0.1)

# this function uses sapply and returns the RMSE of the regularised
# movie + user effects model for each value of lambda in lambdas
tune_lambda_u <- sapply(lambdas, function(lambda){

  # compute mu_hat as the mean of rating in the train_set
  mu_hat <- mean(train_set$rating)

  # compute b_i_hat with train_set, group_by movieID
  # calculation using each value of lambda in lambdas
  reg_movie_effects <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i_hat = sum(rating - mu_hat)/(n()+lambda))

  # compute b_u_hat with train_set, group_by userID
  # left join with b_i_hat
  # calculation using each value of lambda in lambdas
  reg_user_effects <- train_set %>%
    group_by(userId) %>%
    left_join(reg_movie_effects, by = "movieId") %>%
    summarize(b_u_hat = sum(rating - mu_hat - b_i_hat)/(n()+lambda))

  # predict ratings against the test set
  # using calculated b_u_hat, b_i_hat and mu_hat
  predicted_ratings <- test_set %>%
    left_join(reg_movie_effects, by = "movieId") %>%
    left_join(reg_user_effects, by = "userId") %>%
    mutate(prediction = mu_hat + b_i_hat + b_u_hat) %>%
    pull(prediction)

# find and return RMSE using caret::RMSE with predicted ratings and the test_set rating
  return(RMSE(predicted_ratings, test_set$rating))

})
```

We can visualise the value of lambda that provides the lowest RMSE for the regularised movie and user effects model in figure 11.

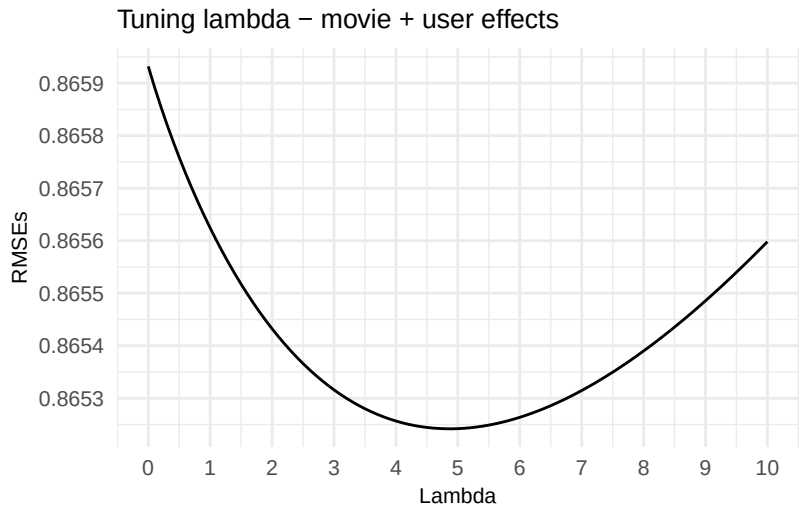

Figure 11: Tuning lambda to minimise the RMSE for regularised movie + user effects

```
# the best lambda is the one that minimises RMSE in the function tune_lambda_u
best_lambda <- lambdas[which.min(tune_lambda_u)]
# print best lambda
best_lambda
```

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

```
# the RMSE of regularised movie + user effects is the minimal RMSE from tune_lambda_u
regularised_rmse_i_u_effects <- min(tune_lambda_u)
# print RMSE
regularised_rmse_i_u_effects
```

```
## [1] 0.8652418
```

Here we note that the RMSE for the regularised movie + user effects model is less than that of the RMSE for the movie + user + genre effects model (table 5, iteration 4: 0.865594). Therefore this is the best movie reccommendation algorithm so far. It is likely we will improve accuracy of the model even further by regularising the movie + user + genre effects model

Regularisation of movie + user + genre effects

We can improve the model even more by regularising the model with all the effects, movie + user + genre. To do this we will use the following formula to find the estimated regularised genre effects parameter, $\hat{b_g}(\lambda)$, and add this to the already establised algorithm.

$$\hat{b_g}(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} \left(Y_{u,i} - \hat{\mu} - \hat{b_i} - \hat{b_u} \right)$$

Once again we will need to tune the lambda parameter with the following function.

```r
# cross validation to tune for which value of lambda to use to regularise the
# movie + user + genre effects model

# define lambdas
lambdas <- seq(0, 10, 0.1)

# this function uses sapply and returns the RMSE of the regularised
# movie + user effects model for each value of lambda in lambdas
tune_lambda_all <- sapply(lambdas, function(lambda){

  # compute mu_hat as the mean of rating in the train_set
  mu_hat <- mean(train_set$rating)

  # compute b_i_hat with train_set, group_by movieID
  # calculation using each value of lambda in lambdas
  reg_movie_effects <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i_hat = sum(rating - mu_hat)/(n()+lambda))

  # compute b_u_hat with train_set, group_by userID
  # left join with b_i_hat
  # calculation using each value of lambda in lambdas
  reg_user_effects <- train_set %>%
    group_by(userId) %>%
    left_join(reg_movie_effects, by = "movieId") %>%
    summarize(b_u_hat = sum(rating - mu_hat - b_i_hat)/(n()+lambda))

  # compute b_g_hat with train_set, group_by genres
  # left join with b_i_hat and b_u_hat
  # calculation using each value of lambda in lambdas
  reg_genre_effects <- train_set %>%
    group_by(genres) %>%
    left_join(reg_movie_effects, by = "movieId") %>%
    left_join(reg_user_effects, by = "userId") %>%
    summarise(b_g_hat = sum(rating - mu_hat - b_i_hat - b_u_hat)/(n()+lambda))

  # predict ratings against the test set
  # using calculated b_g_hat, b_u_hat, b_i_hat and mu_hat
  predicted_ratings <- test_set %>%
    left_join(reg_movie_effects, by = "movieId") %>%
    left_join(reg_user_effects, by = "userId") %>%
    left_join(reg_genre_effects, by = "genres") %>%
    mutate(prediction = mu_hat + b_i_hat + b_u_hat + b_g_hat) %>%
    pull(prediction)

# find and return RMSE using caret::RMSE with predicted ratings and the test_set rating
  return(RMSE(predicted_ratings, test_set$rating))

})
```

We can visualise the value of lambda that provides the lowest RMSE for the regularised movie and user and genre effects model in figure 12.

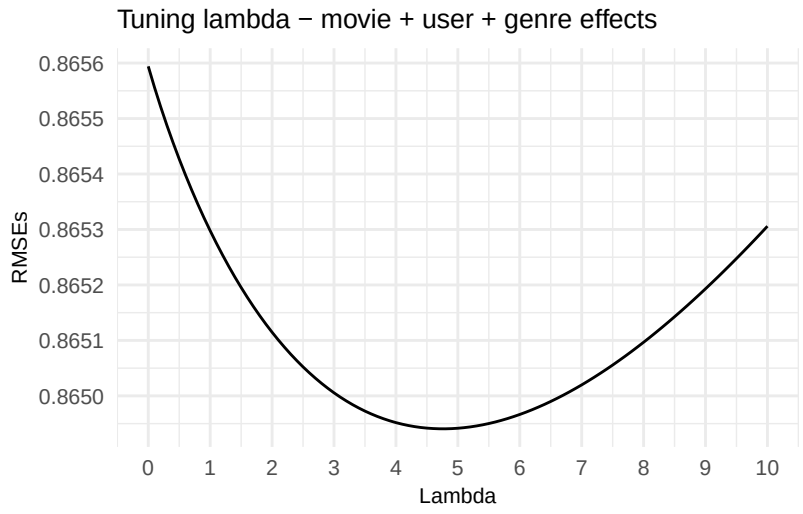

Figure 12: Tuning lambda to minimise the RMSE for regularised movie + user effects

```
# the best lambda is the one that minimises RMSE in the function tune_lambda_all
best_lambda_all <- lambdas[which.min(tune_lambda_all)]
# print best lambda
best_lambda_all
```

```
## [1] 4.8
```

```
# the RMSE of regularised movie + user + genre effects is the minimal RMSE
# from tune_lambda_all
regularised_rmse_all_effects <- min(tune_lambda_all)
#print RMSE
regularised_rmse_all_effects
```

```
## [1] 0.8649406
```

This value of lambda, 4.8, is closer in value to the tuned lambda for the regularised movie + user effects of 4.9, compared with the tuned lambda for the regularised movie effects of 2.4.

Here the RMSE using the tuned lambda of the regularised movie + user + genre effects is the lowest and hence most accurate of all the algorithms so far. We could continue to make further refinements using other machine learning methods. However, we will stop the development at seven model iterations to manage the time and resources needed to calculate the algorithm. The more complex the machine learning algorithm the longer computation time needed. Therefore the regularised movie + user + genre effects will be the final algorithm for the movie recommendation model.

Results

Final model

The improvements of RMSE over the seven models developed in the Methods and analysis section can be found in table 10. Here we can see that the lowest RMSE is for regularised movie + user + genre effects model.

Table 10: RMSE improvements over seven models

method	RMSE
Average baseline	1.0599043
Movie effects	0.9437429
Movie + user effects	0.8659320
Movie + user + genre effects	0.8655941
Regularised movie effects	0.9436745
Regularised movie + user effects	0.8652418
Regularised movie + user + genre effects	0.8649406

With the final algorithm developed we will evaluate it's accuracy against the *final_holdout_test*, using RMSE as the metric. Therefore we will use this algorithm as the final optimised model on the *edx* dataset, from which the final holdout test has been created.

```
# use final regularised algorithm on edx dataset to create the final model
# evaluate the accuracy of final model on the final_holdout_test

# compute mu_hat as the mean of rating in edx
mu <- mean(edx$rating)

# compute b_i with edx,
# group_by movieID
# calculation using tuned lambda from final algorithm
b_i <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu)/(n()+best_lambda_all))

# compute b_u with edx,
# left join with b_i
# group_by userID
# calculation using tuned lambda from final algorithm
b_u <- edx %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - mu - b_i)/(n()+best_lambda_all))

# compute b_g with edx,
# left join with b_i
# left join with b_u
# group_by genres
# calculation using tuned lambda from final algorithm
b_g <- edx %>%
  left_join(b_i, by ="movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(genres) %>%
  summarise(b_g = sum(rating - mu - b_i - b_u)/(n()+best_lambda_all))

# predict final model ratings against the final_holdout_test
# using calculated b_g, b_u, b_i and mu
final_model_ratings <- final_holdout_test %>%
  left_join(b_i, by ="movieId") %>%
  left_join(b_u, by = "userId") %>%
```

```
  left_join(b_g, by = "genres") %>%
  mutate(predictions = mu + b_i + b_u + b_g) %>%
  pull(predictions)

# calculate the final model RMSE using caret::RMSE with final model ratings
# and the final holdout test ratings
rmse_final_model <- RMSE(final_model_ratings, final_holdout_test$rating)
```

The RMSE of the final optimised model shown in table 11 is less than 0.86490.

Table 11: RMSE for the final model

final_method	RMSE
Regularised movie + user + genre effects	0.864451

Model limitations and future optimisation

Another machine learning method we could use to make further improvements to the final optimised model is matrix factorisation. Matrix factorisation as applied to movie recommendation systems is a method that can explain the additional variability related to the existence of similar rating patterns for groups of movies or users.(Irizarry 2019) However, fitting a model using this method is complex and beyond the scope of this project.

Movie reccomendations

We can now apply the final optimised model to make movie reccommendations with the *edx* dataset.

```
# final optimised model used to make predictions with edx dataset
# using, mu, b_i, b_u and b_g already calculated on edx
optimised_model <- edx %>%
  left_join(b_i, by ="movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(predictions = mu + b_i + b_u + b_g)
```

The ten best rated movies using the optimised movie recommendation model are shown in table 12.

```
# best 10 movies using final optimised model
best_ten_movies_reg <- optimised_model %>%
  arrange(desc(predictions)) %>%
  group_by(title) %>%
  distinct(title) %>%
  head(10) %>%
  pull(title)
```

Table 12: Ten best rated movies using optimised model

best_ten	movie_titles
1	Shawshank Redemption, The (1994)
2	Schindler's List (1993)
3	Usual Suspects, The (1995)
4	Seven Samurai (Shichinin no samurai) (1954)
5	Godfather, The (1972)
6	Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
7	Memento (2000)
8	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
9	Star Wars: Episode V - The Empire Strikes Back (1980)
10	Goodfellas (1990)

We can also use the optimised movie recommendation model to choose genre-specific movie picks. The ten best science fiction movies are shown in table 13. The top two Star Wars movies here also feature in the overall top movie list.

```
# 10 best science fiction movies using final optimised model
best_ten_scifi <- optimised_model %>%
  filter(str_detect(genres, "Sci-Fi")) %>%
  arrange(desc(predictions)) %>%
  group_by(title) %>%
  distinct(title) %>%
  head(10) %>%
  pull(title)
```

Table 13: Ten best rated science fiction movies

best_ten	science_fiction
1	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
2	Star Wars: Episode V - The Empire Strikes Back (1980)
3	Matrix, The (1999)
4	Blade Runner (1982)
5	Alien (1979)
6	Clockwork Orange, A (1971)
7	Aliens (1986)
8	Star Wars: Episode VI - Return of the Jedi (1983)
9	Terminator, The (1984)
10	Terminator 2: Judgment Day (1991)

We can also look at the table 14 to see the top ten rated movies in the drama genre. Noting that the top two rated movies are the same as for the top ten overall movie recommendations.

```
# 10 best drama movies using final optimised model
best_ten_drama <- optimised_model %>%
  filter(str_detect(genres, "Drama")) %>%
  arrange(desc(predictions)) %>%
  group_by(title) %>%
  distinct(title) %>%
  head(10) %>%
  pull(title)
```

Table 14: Ten best rated drama movies

best_ten	drama
1	Shawshank Redemption, The (1994)
2	Schindler's List (1993)
3	Seven Samurai (Shichinin no samurai) (1954)
4	Godfather, The (1972)
5	Memento (2000)
6	Goodfellas (1990)
7	Fight Club (1999)
8	American History X (1998)
9	Taxi Driver (1976)
10	Pulp Fiction (1994)

Conclusion

I have developed a movie recommendation model using machine learning approaches on the MovieLens 10M dataset. This was achieved by making iterative improvements over seven algorithms, using linear models and regularisation. I evaluated the accuracy of the algorithms using the RMSE. I was able to get a final model with an RMSE of less than 0.86490, as measured against the final holdout test. Although further improvements could have been made to the algorithm, by adding in more parameters to the linear model, or using matrix factorisation, it was decided to stop development due to time and resource constraints. In the end, the movie recommendation model was able to recommend a believable list of top ten rated movies, as well as a couple of top ten rated genre specific picks.

References

GroupLens. 2025a. "MovieLens 10M Dataset." https://grouplens.org/datasets/movielens/10m/.

———. 2025b. "MovieLens Dataset." https://grouplens.org/datasets/movielens/.

Irizarry, Rafael A. 2019. "Introduction to Data Science: Data Analysis and Prediction Algorithms with r." In, 609–65. https://leanpub.com/datasciencebook. https://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#recommendation-systems.

Jonatasv. 2024. "Metrics Evaluation: MSE, RMSE, MAE and MAPE." Medium. https://medium.com/@jonatasv/metrics-evaluation-mse-rmse-mae-and-mape-317cab85a26b.

Motefaker, Amir. 2024. "Movie Recommendation System Using r - BEST." https://www.kaggle.com/code/amirmotefaker/movie-recommendation-system-using-r-best#Genres.