Movie Recommendation System

Maike Heinrich

24 9 2021

Building a movie recommendation system based on machine learning

1. Introduction / overview

Harvard University launched a challenge to write the code for a movie recommendation system. Fully awarded with points is that final code, that achieves an RMSE (residual or root mean squared error) of less than 0.86490. Years ago, Netflix rewarded a team of data scientists with one million dollars as they achieved an RMSE of about 0.857.

The RMSE is a common used measure of the distance between the predicted values and the actual ones. It's the square root of the average of all distances between these values, squared. So the formula would look like this: RMSE = square root (average (predicted values - actual values) squared).

So let's find out if it's possible to get the full points of Harvard University for this challenge.

Before writing an algorithm to predict ratings, the data set has to be observed and pre-processed. This includes visualizing some interesting facts that help to understand the approach that has to be used for writing the code.

The data set is divided into an "edx"-set to train the machine learning algorithm on and a "validation" set, which will only be used to test the final code, to see the value of the RMSE. So it can't be touched during building the algorithm. Therefore the edx data has to be separated again into two parts, a training and a test set.

The data has to be cleaned from "noisy" data, which means for example users, who only rated one or two movies, or movies which only get one rating - taking these kind of values fully into account can hardly be a representative approach for a solid prediction, because we don't know if this user is very cranky or loves every movie he watches. Therefor regularization will be taken into account and a term will be included which will shrink these insecure estimate of these values towards zero, gives them less weight. This term will be calculated using cross-validation.

Additionally the approach here will be to modify the training data a bit by filtering for users who only rated 20 or more times. This will improve the algorithm as it will be more stable. At the end this algorithm will be used with the complete unchanged edx set with all users and tested on the untouched validation set.

Some movies are similar, so there is a movie to movie effect, likewise some users rate similar, have similar preferences, that's the user to user effect. Both can be taken into account with matrix factorization. This method and how regularization and cross-validation works will be further explained in the methods / analysis part.

So let's get started.

2. Methods / analysis (contains 3 parts)

Part 1: The basic code provided by Harvard University to build the algorithms on (The provided code can be observed in the script file but is not shown here.)

Part 2: Observation of the dataset (dimension, best movies, most rated movie, user-effect, etc) The basic data provided by Harvard University is the "movielens" data set, where almost 70,000 users rated over 10,000 movies.

```
# How many movies are in the edx dataset
length(unique(edx$movieId))
```

```
## [1] 10677
```

```
# How many users rated movies
length(unique(edx$userId))
```

```
## [1] 69878
```

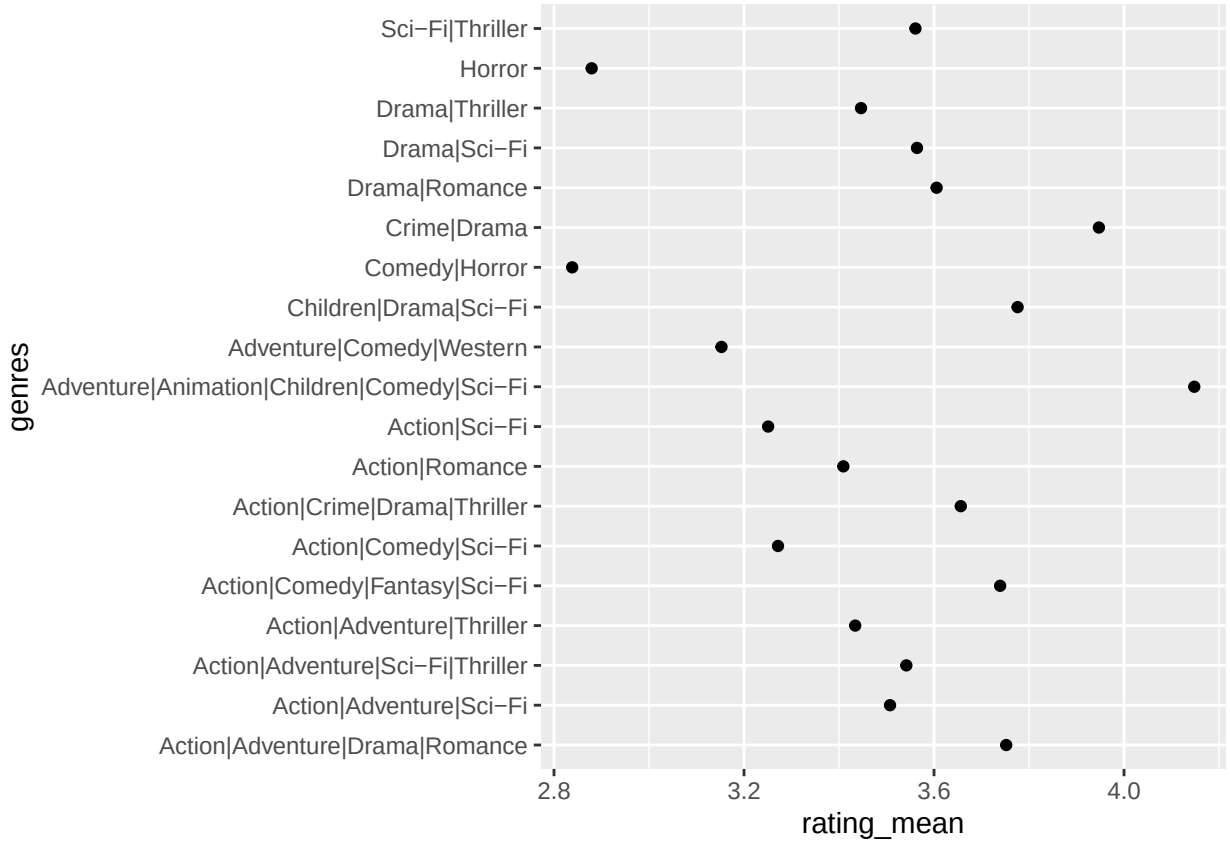
To produce an action movie seems to be a good approach for getting high ratings, as a look at the ten best rated genres shows. To make sure to get a plausible result, only genres are kept, which are rated more than 100 times.

```
# 10 best rated genres which are rated more than 100 times
genres <- edx %>% group_by(genres) %>%
  filter(table(genres) > 100) %>%
  arrange(desc(rating))

genres$genres[1:10]
```

```
##  [1] "Comedy|Romance"
##  [2] "Action|Crime|Thriller"
##  [3] "Action|Drama|Sci-Fi|Thriller"
##  [4] "Action|Adventure|Sci-Fi"
##  [5] "Action|Adventure|Drama|Sci-Fi"
##  [6] "Children|Comedy|Fantasy"
##  [7] "Comedy|Drama|Romance|War"
##  [8] "Adventure|Children|Romance"
##  [9] "Adventure|Animation|Children|Drama|Musical"
## [10] "Action|Comedy"
```

Let's take a look at 20 randomly selected genres and show how they are rated on average.

```
# Sample 20 random genres and plot them against the ratings
set.seed(11, sample.kind = "Rounding")

random_genres <- sample(genres$genres, 20)

ind<- edx %>% filter(genres %in% random_genres) %>%
  group_by(genres) %>%
  summarize(rating_mean = mean(rating))

genre_rating <- (unique(ind))

genre_rating %>% ggplot(aes(rating_mean, genres))+
  geom_point()
```



Some of the best rated movies are “Forrest Gump”, “The Net” and “Stargate” with fully five points.

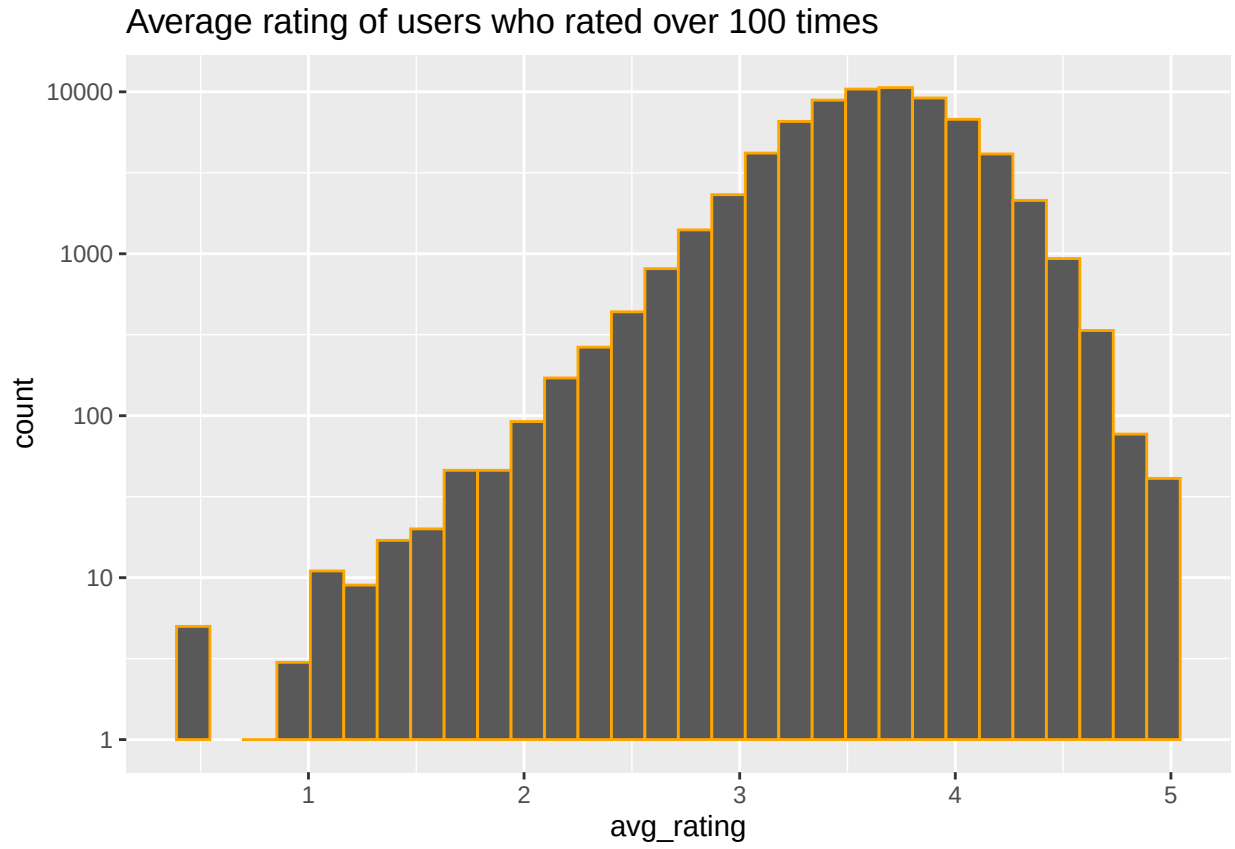
```
# 10 of the best rated movies
best_ratings <- edx %>% group_by(rating) %>%
  arrange(desc(rating))

head(best_ratings, 10)
```

```
## # A tibble: 10 x 6
## # Groups:   rating [1]
##    userId movieId rating timestamp title                 genres
##     <int>   <dbl>  <dbl>     <int> <chr>                 <chr>
##  1      1     122      5 838985046 Boomerang (1992)      Comedy|Romance
##  2      1     185      5 838983525 Net, The (1995)       Action|Crime|Thriller
##  3      1     292      5 838983421 Outbreak (1995)       Action|Drama|Sci-Fi|T~
##  4      1     316      5 838983392 Stargate (1994)       Action|Adventure|Sci-~
##  5      1     329      5 838983392 Star Trek: Generation~ Action|Adventure|Dram~
##  6      1     355      5 838984474 Flintstones, The (199~ Children|Comedy|Fanta~
##  7      1     356      5 838983653 Forrest Gump (1994)   Comedy|Drama|Romance|~
##  8      1     362      5 838984885 Jungle Book, The (199~ Adventure|Children|Ro~
##  9      1     364      5 838983707 Lion King, The (1994)  Adventure|Animation|C~
## 10      1     370      5 838984596 Naked Gun 33 1/3: The~ Action|Comedy
```

To see how users, who rated over 10 movies vote on average, take a look at the following plot. It’s in log-10 scale to illustrate the distribution better. It shows, that on the very left side there are users who generally on average rate movies very low while on the right side are some users, who gave every movie they watched,

fully five points on average. It shows, that there is a user effect, that has to be taken into account. If now a very cranky user rates a very good movie, there will be added a term, "user_effect" which regulates this ranking. Later I'll show, how to use it.

```
# Histogram of distribution of user ratings to show the user effect
edx %>% group_by(userId) %>%
  summarize(avg_rating = mean(rating)) %>%
  filter(n() > 10) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(bins = 30, color = "orange") +
  scale_y_log10() +
  ggtitle("Average rating of users who rated over 100 times")
```



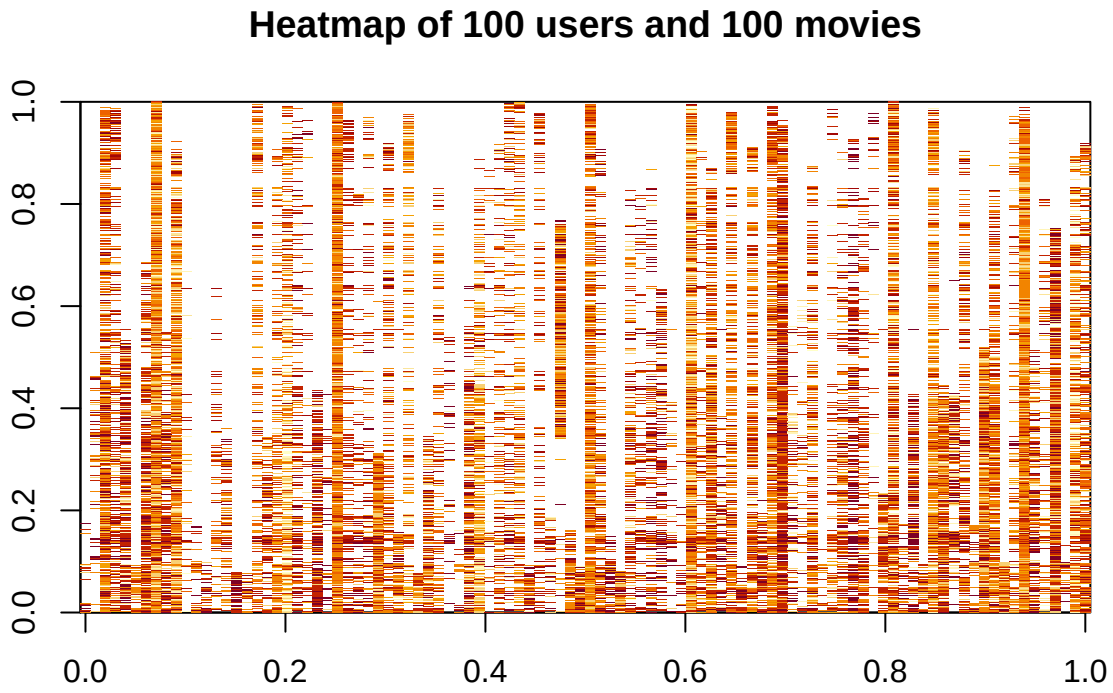
The overall average of all ratings is about 3.5

```
# Overall rating average
mean(edx$rating)
```

```
## [1] 3.512465
```

To see, what a recommendation system actually is meant for, here's an image of a random subset of the data set of 100 users (on the x axis) and 100 movies (on the y axis). It shows clearly how many values are missing (all the white cells). This is because not every user saw and rated every movie, not surprisingly. To predict all these missing values is the task of the machine learning algorithm of the recommendation system. And this system is gonna be built in the following section.

```
# Heatmap of a random subset with 100 users and movies to see the missing values
edx_sample_ind<- sample(edx$userId, 100)
edx_sample <- edx %>% filter(userId %in% edx_sample_ind)

user_movie_matrix <- as.matrix(dcast(edx_sample, userId~movieId, value.var = "rating", na.rm = FALSE)[,
image(user_movie_matrix, main = "Heatmap of 100 users and 100 movies")
```



Part 3: The recommendation system (the training procedure) To start with the training procedure the provided "edx" data set, which is 90 percent of the whole "movielens" data set will be separated into a training and a test set. The training data will be 80 percent of "edx", the test set 20 percent.

```
# Create a data partition of the edx set to have a train and a test set
set.seed(7, sample.kind = "Rounding")

test_index <- createDataPartition(edx$rating, times = 1, p = 0.8, list = FALSE)
train_set <- edx %>% slice(test_index)
test_set <- edx %>% slice(-test_index)
```

Now to improve the precision of the algorithm, with the following code only users remain in the train set, who rated more than 20 times. This is only used for building our training algorithm. Later the complete and unchanged edx set with all the users in it and the complete validation set will be used to test the algorithm.

```
# Only keep users with over 20 ratings to avoid noisy estimates.
# This is only for training the algorithm, later the complete edx set
```

```
# and the complete validation set will be used for testing.
train_set <- train_set %>% group_by(userId) %>% filter(table(userId) > 20)
```

It's important to make sure that all movies and users, who are in the test set are also in the train set, otherwise the code will produce NAs, "not available" values. This is possible by semi-joining the test set by the train set.

```
# Make sure that all movies and users in the test set are also in the train set to avoid NAs
test_set <- test_set %>%
  semi_join(train_set, "userId") %>%
  semi_join(train_set, "movieId")
```

Now matrix factorization will be used to decompose the data matrix into lower dimensional products. This will be achieved by grouping the data for movies and users to find the specific effects. Matrix factorization is a collaborative method as the decomposed matrices calculations will be build onto each other.

To calculate the movie and user effects we first need to define the average of all ratings through all movies. By the way, if only this average would be used for our predictions, the RMSE would be around 1, so quite far from the desired result.

```
# Define the average of all ratings
rating_avg <- mean(train_set$rating)
```

The movie effect is now defined as the average distance of each movie rating to the average of all movies. So once the data is grouped by movies, the average rating for every movie can be calculated. Some movies like blockbusters get on average higher ratings than others. This is the movie effect.

```
# Define the movie to movie effect, the average distance each movie has to the average of all movies
movie_effect <- train_set %>% group_by(movieId) %>%
  summarize(movie_effect = mean(rating - rating_avg))
```

Now the user effect is defined as the average distance of each user rating to the average of all ratings after removing the movie effect. As shown before, there are some users who's average rating is pretty low while others give 5 points to every movie. So the average rating of every user, as the data is grouped by user, will help to find the user effect here.

```
# Define user to user effect, the distance each user rates from the average of all users
user_effect <- train_set %>%
  left_join(movie_effect, by = "movieId") %>%
  group_by(userId) %>%
  summarize(user_effect = mean(rating - rating_avg - movie_effect))
```

To get an idea what this effect will cause on the ratings, when it will be joined to the test set, the first few lines of the table are shown here.

```
# See how this effect looks like
head(user_effect)
```

```
## # A tibble: 6 x 2
##   userId user_effect
##    <int>       <dbl>
## 1      3     0.168
```

```
## 2      4     0.634
## 3      5     0.00694
## 4      6     0.352
## 5      7    -0.00442
## 6      8     0.192
```

Now these algorithm based on matrix factorization will be tested on the test set. The RMSE will be quite low as the data was modified earlier, it will rise a bit when it will be used later on the final validation set.

```
# Predict the ratings on the test set
predictions <- test_set %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  mutate(predictions = rating_avg + movie_effect + user_effect)

# Look at the prediction table
head(predictions)
```

```
##    userId movieId rating  timestamp
## 1:      3    1252    4.0 1133571071
## 2:      3    1564    4.5 1136418605
## 3:      3    3408    4.0 1164885590
## 4:      3    4535    4.0 1164885526
## 5:      3    5527    4.5 1164885648
## 6:      3    6539    5.0 1133571238
##                                                        title
## 1:                                             Chinatown (1974)
## 2:                         For Roseanna (Roseanna's Grave) (1997)
## 3:                                      Erin Brockovich (2000)
## 4:                                Man From Snowy River, The (1982)
## 5:                                             Possession (2002)
## 6: Pirates of the Caribbean: The Curse of the Black Pearl (2003)
##                                genres movie_effect user_effect predictions
## 1: Crime|Film-Noir|Mystery|Thriller   0.73153055   0.1677439    4.409141
## 2:             Comedy|Drama|Romance   0.05990070   0.1677439    3.737511
## 3:                            Drama   0.09739718   0.1677439    3.775008
## 4:            Drama|Romance|Western   0.18028986   0.1677439    3.857900
## 5:                  Drama|Romance  -0.21318625   0.1677439    3.464424
## 6:  Action|Adventure|Comedy|Fantasy   0.38316275   0.1677439    4.060773
```

```
# Calculate the RMSE on the test set
RMSE(test_set$rating, predictions$predictions)
```

```
## [1] 0.8630293
```

To further improve this algorithm an important step is the regularization. Still there are some obscure movies getting high average ratings because they were only rated by one or two users. These movies distort the prediction as they give large estimates from only small sample sizes. To give these kind of ratings less weight, a term will be added called "lambda", which will regulate these misleading estimates.

To find this term lambda a function is going to help. It calculates the whole procedure of matrix factorization with the regulated movie and user effects (reg_movie_effect, reg_user_effect) and tests a sequence of 51 different values of lambda from 1 to 6 in a distance of 0.1 which will be added to the matrix factorization

terms. Then it will be tested on the test set. This procedure is called cross-validation, because we test the lambda on a separated test set, in this case 51 times, to finally get the desired result. So we validate it through testing across different sets of data. This one lambda, which achieves the lowest value of RMSE, means helps to make the best predictions, will be chosen to build the further algorithm.

```
# Using regularization term lambda to shrink the estimates when movies have only few ratings
# and train the new regulated movie and user effects.
# Find the best lambda using cross-validation
lambdas <- seq(1, 6, 0.1)
rmses <- sapply(lambdas, function(lambda){
  rating_avg <- mean(train_set$rating)
  reg_movie_effect <- train_set %>%
    group_by(movieId) %>%
    summarize(reg_movie_effect = sum(rating - rating_avg)/(n()+lambda))
  reg_user_effect <- train_set %>%
    left_join(reg_movie_effect, by="movieId") %>%
    group_by(userId) %>%
    summarize(reg_user_effect = sum(rating - reg_movie_effect - rating_avg)/(n()+lambda))
  predictions <- test_set %>%
    left_join(reg_movie_effect, by = "movieId") %>%
    left_join(reg_user_effect, by = "userId") %>%
    mutate(predictions = rating_avg + reg_movie_effect + reg_user_effect) %>%
    .$predictions
  return(RMSE(predictions, test_set$rating))
})

RMSE <- min(rmses)
RMSE
```

```
## [1] 0.8625616
```

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

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

Results

Final training and testing Now the algorithm is ready to train it on the whole edx set.

```
# Final training with the regulated algorithm on the complete edx set to get all users and
# movies back
reg_movie_effect <- edx %>%
  group_by(movieId) %>%
  summarize(reg_movie_effect = sum(rating - rating_avg)/(n()+lambda))
reg_user_effect <- edx %>%
  left_join(reg_movie_effect, by="movieId") %>%
  group_by(userId) %>%
  summarize(reg_user_effect = sum(rating - rating_avg - reg_movie_effect)/(n()+lambda))
```

Let's see which RMSE this algorithm can reach when it is tested on the validation set.

```
# Left-join the users and movies of the validation set just to get the same users and movies
# The ratings of the validation set are not used or even touched
final_predictions <- validation %>%
  left_join(reg_movie_effect, by = "movieId") %>%
  left_join(reg_user_effect, by = "userId") %>%
  mutate(predictions = rating_avg + reg_movie_effect + reg_user_effect)

# Now the final-predictions table has the same length as the validation set
dim(final_predictions)
```

```
## [1] 999999      9
```

```
dim(validation)
```

```
## [1] 999999      6
```

```
# Final test on the validation set
RMSE(final_predictions$predictions, validation$rating)
```

```
## [1] 0.86482
```

In usage: Making predictions Now as we have a machine learning recommendation algorithm, let's try it to make some predictions for some of the users. Two users will be randomly picked from the validation set and shown which movies they liked the most. Then the algorithm of the final predictions will recommend some movies for these users.

```
# Picking user 212 randomly and see which 5 movies he liked most
user_212 <- validation %>% filter(userId == 212)
Top_user_212 <- user_212 %>% filter(rating == 5) %>% select(title)
Top5_user_212 <- Top_user_212[1:5]
Top5_user_212
```

```
##                           title
## 1:           Taxi Driver (1976)
## 2:       Pather Panchali (1955)
## 3:          Paris, Texas (1984)
## 4:                  Rope (1948)
## 5: Mirror, The (Zerkalo) (1975)
```

```
# Show movies, the algorithm recommends for this user
Recommend_user_212 <- final_predictions %>%
  filter(userId == 212) %>%
  anti_join(Top5_user_212, by = "title") %>%
  arrange(desc(predictions)) %>%
  select(title)
head(Recommend_user_212)
```

```
##                                                    title
## 1:                                    Fight Club (1999)
## 2:                                     Duck Soup (1933)
```

```
## 3:                                                 Oldboy (2005)
## 4:                                      This Is Spinal Tap (1984)
## 5: Passion of Joan of Arc, The (La Passion de Jeanne d'Arc) (1928)
## 6:                                       To Catch a Thief (1955)
```

```
# Picking user 1724 randomly and see which 5 movies he liked most
user_1724 <- validation %>% filter(userId == 1724)
Top_user_1724 <- user_1724 %>% filter(rating == 5) %>% select(title)
Top5_user_1724 <- Top_user_1724[1:5]
Top5_user_1724
```

```
##                                                    title
## 1: Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
## 2:                                                Fargo (1996)
## 3:                                   Some Like It Hot (1959)
## 4:                                Maltese Falcon, The (1941)
## 5:                                          Ninotchka (1939)
```

```
# Show movies, the algorithm recommends for this user
Recommend_user_1724 <- final_predictions %>%
  filter(userId == 1724) %>%
  anti_join(Top5_user_1724, by = "title") %>%
  arrange(desc(predictions)) %>%
  select(title)
head(Recommend_user_1724)
```

```
##                                          title
## 1:                             Fight Club (1999)
## 2:                 Manchurian Candidate, The (1962)
## 3:                           Sixth Sense, The (1999)
## 4:                                 Manhattan (1979)
## 5:                                   Charade (1963)
## 6: Wings of Desire (Der Himmel Ã¼ber Berlin) (1987)
```

Conclusion

A movie recommendation system based on a machine learning algorithm can help to make relatively precise predictions for users. In this case mainly matrix factorization and regularization is used to train the algorithm. The innovation here was to modify the training set to improve the precision without touching the final validation set, which would have distorted the results. The task of Harvard University to reach an residual mean squared error (RMSE) of less than 0.86490 to get the top points could be achieved. The trained algorithm reached an RMSE of 0.86482.

The methods used here are not the only ones to train a recommendation model. Further techniques are already in use and will help to improve artificial intelligence systems even more in future.

Nevertheless these recommendation systems have their limits and will probably always have. Human minds are unique and even if an algorithm can make a very precisely prediction, it's never a guarantee for really reaching the user's taste. But still it remains a very useful tool which helps to facilitate our way of using data.