Movie Recommendation System

Samuel Velez Arango

03/06/2019

Introduction

The Netflix challenge offered $1 million dollars to the team that could provide a movie reccomendation system that improved theirs by 10%. While the Netflix data set isn't public, MovieLens provides a similar one. We are going to attempt to provide a solution to the challenge (through the MovieLens data set) using linear regression, trying out three different approaches.

Each row in the MovieLens database corresponds to the rating one user assigned to a movie, and there's and ID that identifies each individual user.

```
library(dslabs)
library(tidyverse)
data("movielens")
```

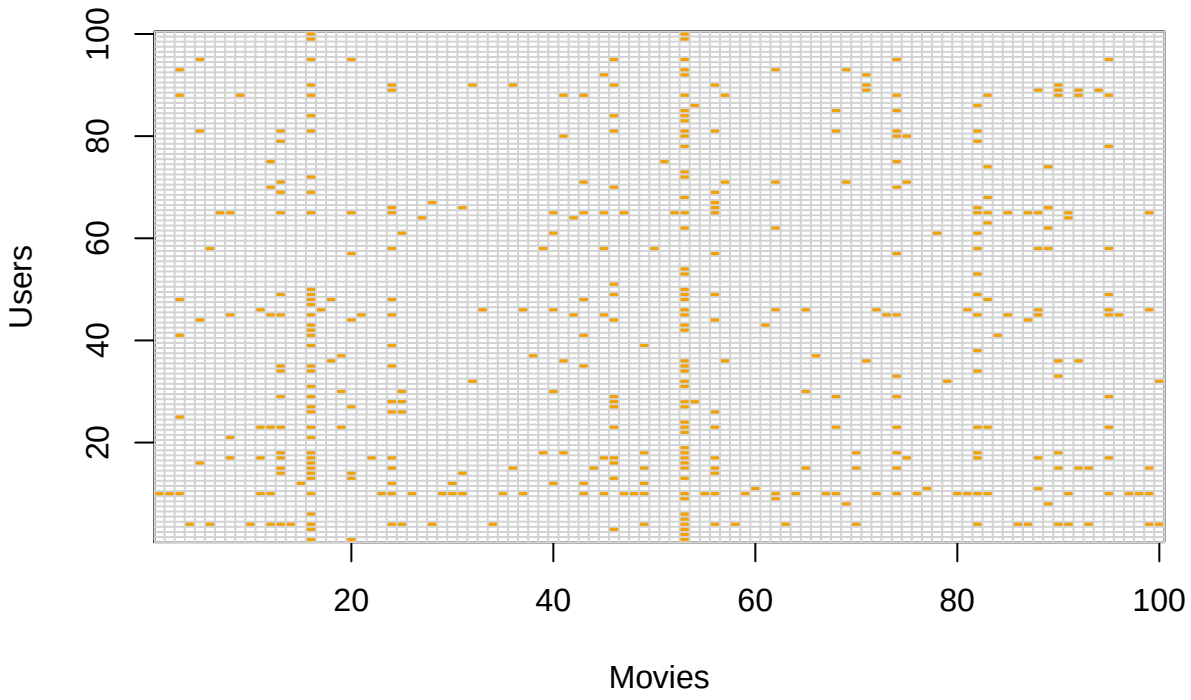
The data contains rating information provided by 671 users on 9066 movies:

```
library(dplyr)
```

```
count_users <-  length(unique(movielens$userId))
count_movies <- length(unique(movielens$movieId))
data.frame(count_users,count_movies)
```

```
##   count_users count_movies
## 1         671         9066
```

However not every user rated every movie, and therefore many fields in the matrix will be empty, as we can see in the following sample plot highlighting the ratings we do have:



Pre-processing

We are going to partition the data into a training and a test data subset, the first containing 80% of the data and the latter the remaining.

```
library(caret)
```

```
set.seed(1)
data_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.8, list = FALSE)
test_set <- movielens[-data_index,]
train_set <- movielens[data_index,]
```

To make sure we don't have users or movies in one set that don't appear in the other we use the semi_join function

```
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

The winners of the challenge were defined based on the Residual Sum of Squared Errors, RMSE. To win, the resulting RMSE had to be lower than 0.85. The RMSE was stated like this:

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

Recommendation system

To create the recommendation system, we will use an iterative process, increasing on the complexity of the previous attempt until the data set is fully exploited.

Simplest solution: disregarding movie ratings and user preferences.

The simplest possible solution is to suggest the same movie for all users, regardless of their preferences. This would be represented by a model that chalks up all variations in preferences to the standard error:

```
average_rating <- mean(train_set$rating)
average_rating
```

```
## [1] 3.544285
```

Through the previous code, our model will only use the average rating and an error term. The RMSE can be calculated as follows:

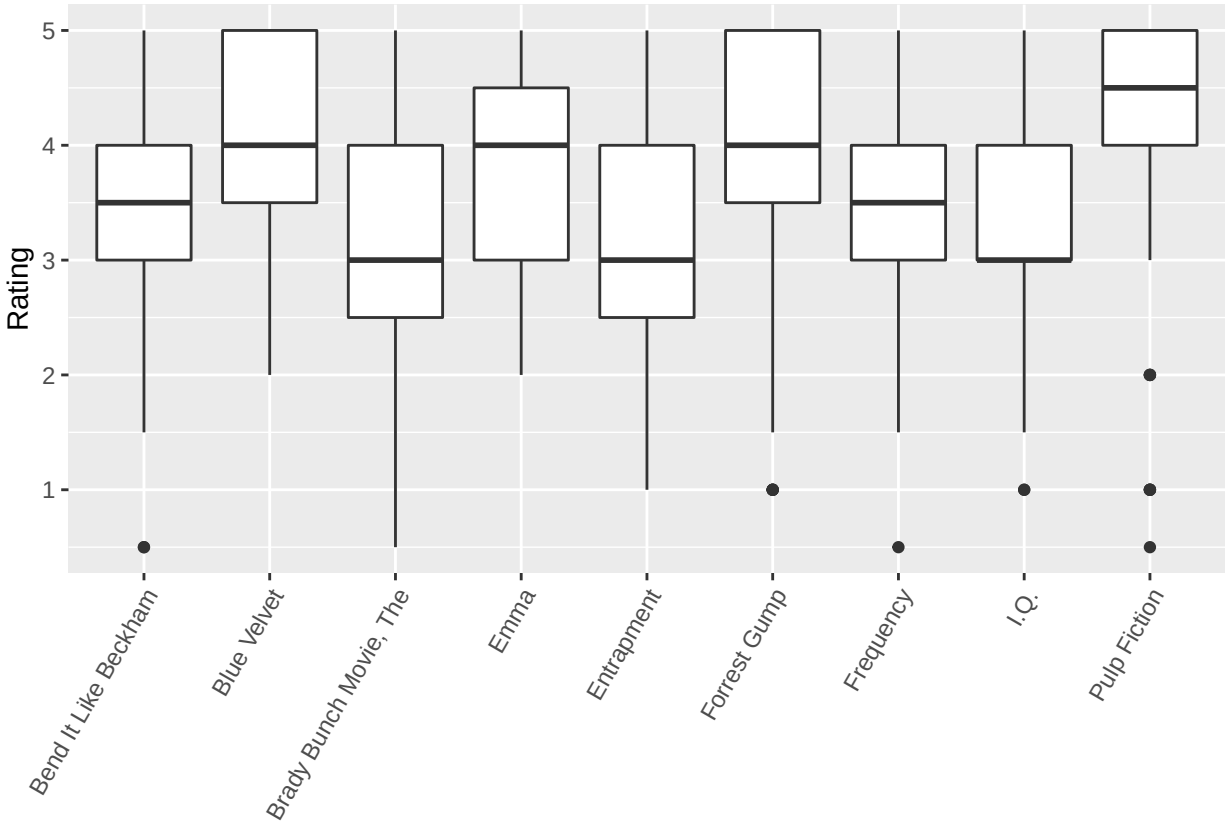
```
simple_rmse <- RMSE(test_set$rating,average_rating)
simple_rmse
```

```
## [1] 1.054886
```

And so, the simplest possible solution provides an RMSE of a little over 1.

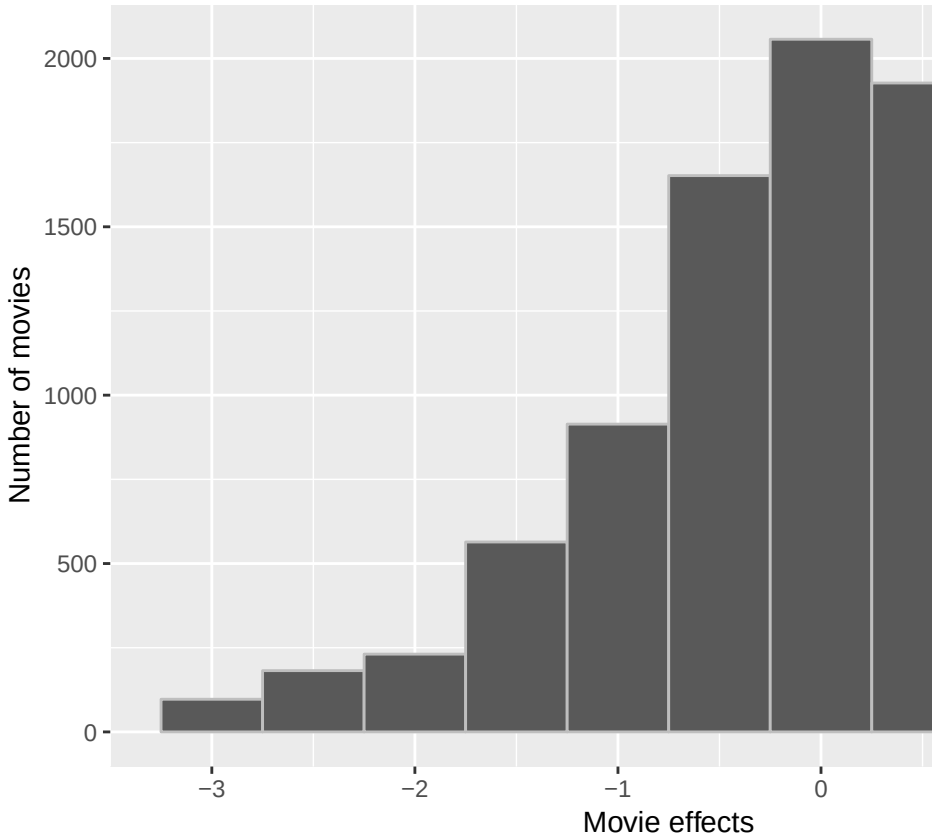
Recommendation based on movie ratings

Another approach would imply taking into account movie ratings. Some movies have a better average rating than others:



And we can use this for the recommendation system by accounting for individual movie ratings. In the Netflix challenge, the difference between the overall average rating and the average rating of each movie was called the bias, and we will implement it with the following code, that groups by movie and takes the average of its ratings, to later substract the total average:

```
movie_average <- train_set %>%
  group_by(movieId) %>%
  summarize(movie_effects = mean(rating - average_rating))
```



And we can see the distribution of this bias:

The expressed model would be of the form Prediction = Overall Movie Average + Movie Bias + Random Error, and its RMSE is the following:

```
predicted_ratings <- average_rating + test_set %>%
  left_join(movie_average, by='movieId') %>%
  pull(movie_effects)

movie_effects_rmse <- RMSE(predicted_ratings, test_set$rating)
movie_effects_rmse
```

```
## [1] 0.9882169
```

Taking each movie's average rating into account improves our RMSE slightly.

Recommendation based on individual preference and movie rating

Introducing the individual preference of each user will add the last layer of information provided by the MovieLens data set:

The following code is similar to the one used to find the movie bias; we are now finding each user's preference bias by removing both the movie effect and the mean overall rating effect.

```
user_average <- train_set %>%
  left_join(movie_average, by='movieId') %>%
  group_by(userId) %>%
  summarize(user_effects = mean(rating - average_rating - movie_effects))
```

Afterwards, with the user bias, we create a formula of the form: Prediction = Overall Movie Average + Movie Bias + User Bias + Error, so when a hard to please user who rates all movies below their average rating needs a suggestion, the effect of his manifested preferences will be negated against the movie's rating, and we can also identify correctly when he isn't bashing a movie but giving a personally above average review:

```
predicted_ratings <- test_set %>%
  left_join(movie_average, by='movieId') %>%
  left_join(user_average, by='userId') %>%
  mutate(prediction = average_rating + movie_effects + user_effects) %>%
  pull(prediction)
```

The final results:

```
user_effects_rmse <- RMSE(predicted_ratings, test_set$rating)
user_effects_rmse
```

```
## [1] 0.9088733
```

Conclusion

With each iteration our results got progressively better, and while this result isn't enough to win the Netflix prize, this is a fun exercise that only needs linear models.

```
## # A tibble: 3 x 2
##   method                          RMSE
##   <chr>                          <dbl>
## 1 Average of all ratings       1.05
## 2 Movie Effects Model          0.988
## 3 Movie and User Effects Model 0.909
```