Movie Recommendation model for the Movielens dataset

Vincent

April 7th 2019

Introduction

Recommendations systems are used in many areas (movies, music, books, news, etc.) and are and important branch of machine learning. For this project we will use the [MovieLens 10M dataset] https://grouplens.org/datasets/movielens/10m/ which provides 10 million ratings for 10,000 movies by 72,000 users. From this data our goal is to build a model allowing us to predict which rating a user would give to a given movie, based on what we know on movies and users. We will evaluate this model with the Root Mean Square Error (RMSE) and try to minimize this indicator with a reasonable computation time and memory consumption.

Loading the Data

Loading useful libraries

```r
library(tidyverse)
library(caret)
```

Downloading the data if required

```r
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- read.table(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                      col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
```

Formatting data

```r
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

Splitting the data in a training and a validation test

```r
# Validation set will be 10% of MovieLens data

set.seed(1)
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 validation set are also in edx set
```

```r
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

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

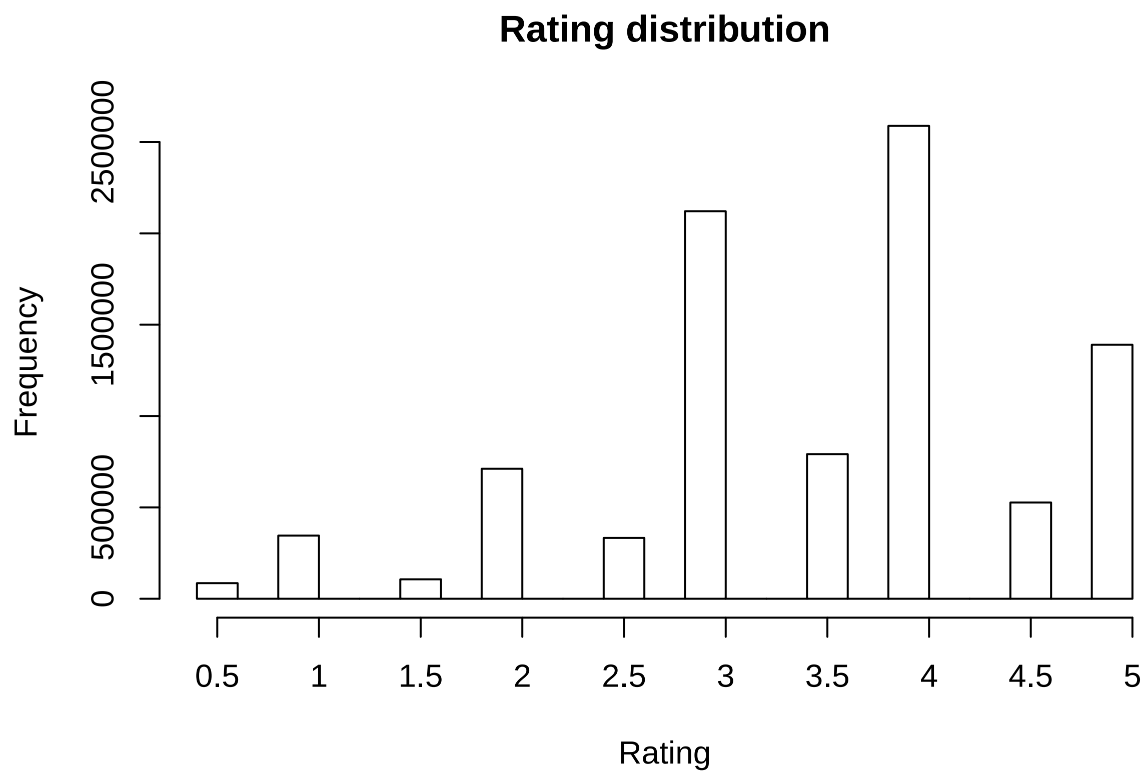
Exploratory analysis

```r
glimpse(edx)
```

```
## Observations: 9,000,055
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 83898339...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (19...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|D...
```

Rating distribution

```r
#Distribution of Movie Ratings
hist(edx$rating, xaxt='n', xlab = "Rating", main = "Rating distribution")
axis(side=1, at=sort(unique(edx$rating)), labels=sort(unique(edx$rating)))
```

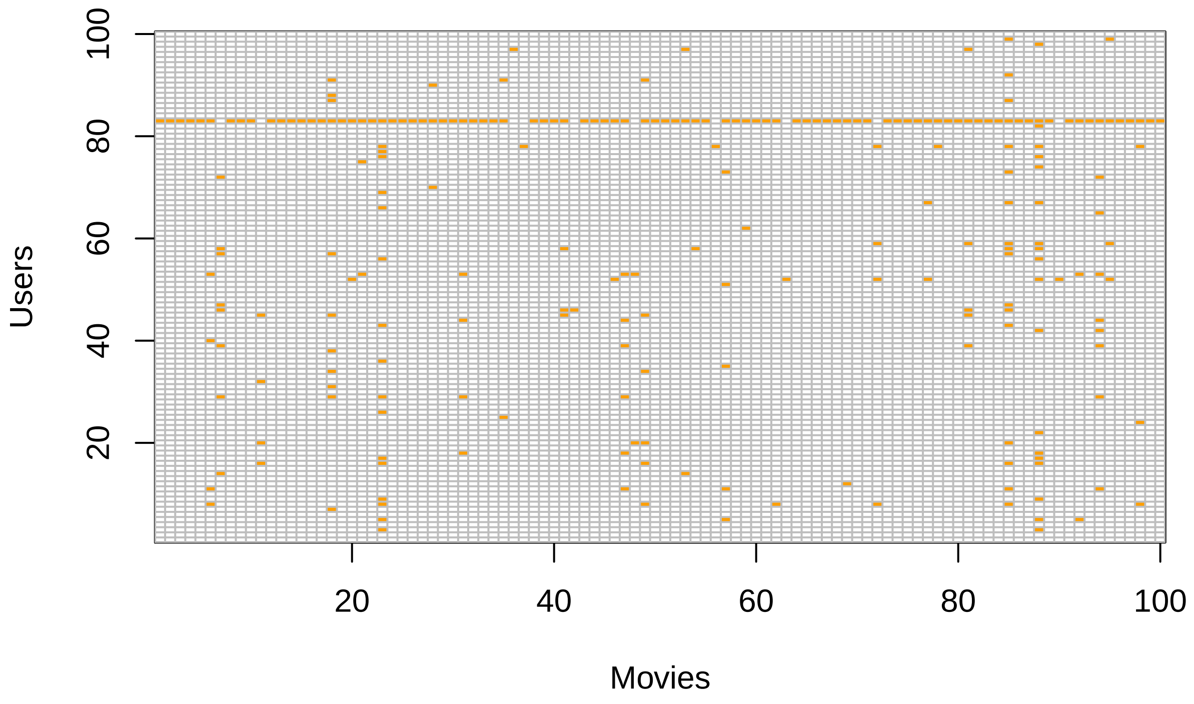


Ratings range from 0.5 to 5 (No 0 rating). The mode is 4 and the mean is between 3 and 4. We also observe that ratings are most likely to be integers. Ratings with a .5 decimal value are less common.

Sparsity

userId	Braveheart (1995)	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Shawshank Redemption
13	NA	NA	NA	4	
16	NA	NA	3	NA	
17	NA	NA	NA	NA	
18	4.5	NA	3	5	
19	5.0	4	1	NA	
22	5.0	NA	4	5	
23	NA	NA	NA	4	
26	5.0	NA	NA	NA	
30	5.0	5	4	NA	

If we organise our data with users as rows and films as columns, we see that the resulting matrix is very sparse, meaning that it contains many missing values because not every user has rated evry movie. We can visualise this sparsity in the image below.



The prediction task consists of filling all these missing values with our predictions, based on what we know about users ans movies from the existing ratings.

Method for building and evaluating recommendation models

Prediction Evaluation with RMSE

We will use the Root Mean Square Error (RMSE) as a loss function to evaluate our predictions.

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

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

Baseline model

To create a baseline, we will create a simplistic model that predict the mean of the ratings in the training set

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

Our baseline model always predicts the mean of all ratings, which is about 3.5

```r
RMSE(validation$rating, rep(mean(edx$rating), nrow(validation)))
```

```
## [1] 1.061202
```

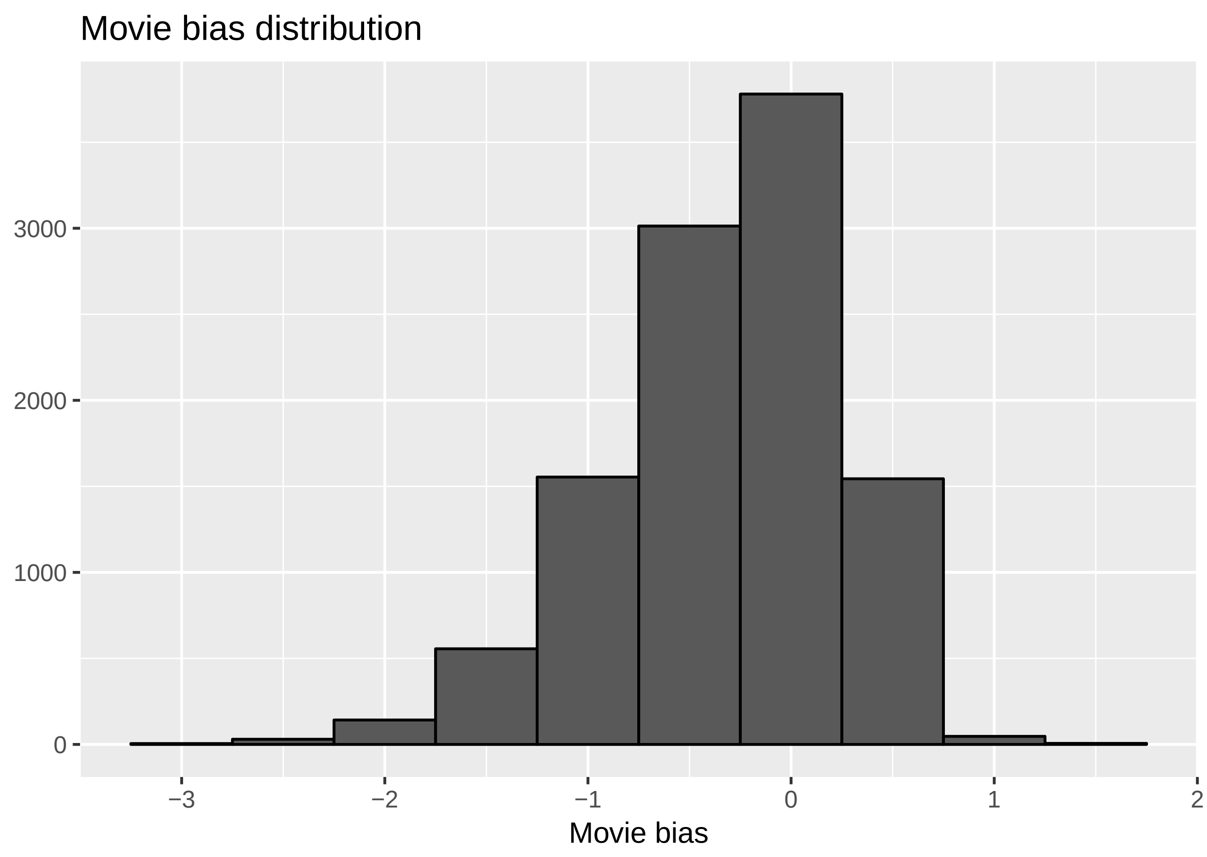
The RMSE shows that with simplistic approach our rating prediction is on average off by one. A more elaborate model should help reducing this error.

Taking into account the movie bias.

Some movies are better than others, even if we disregard any user effect. Taking into account this movie bias should help reduce our errors in prediction.

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

```r
mu <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., xlab = "Movie bias", main = "Movie bi
```



```r
model1_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

model1_rmse <- RMSE(model1_ratings, validation$rating)
model1_rmse
```

```
## [1] 0.9439087
```

There is already an improvement over the baseline model.

Regularization of the movie bias

With regularisation, we want to penalise large estimates that come from small sample sizes, because we are less sure of them and they can lead to large errors. Instead of minimizing the least square equation, we minimize an equation that adds a penalty:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i)^2+\lambda\sum_i b_i^2$$

The values of b_i that minimize this equation are:

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

λ is a tuning parameter that we can optimise using 10-fold validation.

```r
set.seed(1971)
# Randomly shuffle the data
edx_shuffled<-edx[sample(nrow(edx)),]
# Create 10 equally size folds
folds <- cut(seq(1,nrow(edx)),breaks=10,labels=FALSE)
# Perform 10 fold cross validation
bestlambda <-c()
for(i in 1:10){
  # Segment data by fold
  testIndexes <- which(folds==i,arr.ind=TRUE)
  tmp_testData <- edx_shuffled[testIndexes, ]
  trainData <- edx_shuffled[-testIndexes, ]
  # Make sure userId and movieId in test set are also in training set
  testData <- tmp_testData %>%
    semi_join(trainData, by = "movieId") %>%
    semi_join(trainData, by = "userId")
  # Add rows removed from test set back into training set
  removed <- anti_join(tmp_testData, testData)
  trainData <- rbind(trainData, removed)

  lambdas <- seq(0, 10, 0.25)

  mu <- mean(trainData$rating)
  just_the_sum <- trainData %>%
    group_by(movieId) %>%
    summarize(s = sum(rating - mu), n_i = n())

  rmses <- sapply(lambdas, function(l){
    predicted_ratings <- testData %>%
      left_join(just_the_sum, by='movieId') %>%
      mutate(b_i = s/(n_i+l)) %>%
      mutate(pred = mu + b_i) %>%
      pull(pred)
    return(RMSE(predicted_ratings, testData$rating))
  })

  bestlambda <- c(bestlambda, lambdas[which.min(rmses)])
}
mean(bestlambda)
```

```
## [1] 2.15
```

Cross validation leads to a λ value of 2.15. We use this lambda value to evaluate our model on the test set.

```
l <- mean(bestlambda)

mu <- mean(edx$rating)
just_the_sum <- edx %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

  predicted_ratings <- validation %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)

  RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.9438523
```

There is a slight improvement over the non-regularized model.

Taking into account the regularized user bias

Some users are more strict than others. To take this into account we can add a user bias term similar to the movie bias, and once again we want to penalize large bias from small sample. The equation to minimize becomes:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$$

```
bestlambda <-c()
for(i in 1:10){
  #Segment data by fold
  testIndexes <- which(folds==i,arr.ind=TRUE)
  tmp_testData <- edx_shuffled[testIndexes, ]
  trainData <- edx_shuffled[-testIndexes, ]
  # Make sure userId and movieId in testing set are also in training set
  testData <- tmp_testData %>%
    semi_join(trainData, by = "movieId") %>%
    semi_join(trainData, by = "userId")
  # Add rows removed from test set back into training set
  removed <- anti_join(tmp_testData, testData)
  trainData <- rbind(trainData, removed)

  #Use the test and train data
  lambdas <- seq(0, 10, 0.25)
```

```r
  mu <- mean(trainData$rating)

  rmses <- sapply(lambdas, function(l){

  b_i <- trainData %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- trainData %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <-
    testData %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

    return(RMSE(predicted_ratings, testData$rating))
  })

  bestlambda <- c(bestlambda, lambdas[which.min(rmses)])
}
mean(bestlambda)
```

```
## [1] 4.925
```

Cross validation leads to a λ value of 4.925. We use this lambda value to evaluate our model on the test set.

```r
l <- mean(bestlambda)

  mu <- mean(edx$rating)

  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

    final_RMSE <- RMSE(predicted_ratings, validation$rating)
    final_RMSE
```

```
## [1] 0.8648183
```

The RMSE for our last model has improved to 0.8648183.

Possible next steps

Other steps could be used to improve our model. In particular we have not used features like movie genres or rating date. We could also take advantage of libraries available in R like recommenderlab to build our predictions. These libraries usualy implement various algorithms besides the matrix factorization approach we have taken.

Results

A first basic approach (always predicting the mean of predictions) led to a RMSE larger than 1. This result was incrementally improved by including the movie bias in our model, then by regularizing this bias ter (penalizing large bias terms with small sample sizes). Finally the model was further improved by including a regularized user bias in our model. During regularization, a 10-fold cross validation was used to optimize the lambda parameter without using the validation set. The validation set was not used in any way for model building, only for final evaluation. The final RMSE obtained is ***0.8648183***.

Conclusion

The simple approach we took allows to build a recommendation model with reasonable memory and CPU requirements. More elaborate approaches exist and R provides well documented packages to implement these models. However, the dataset size can be a challenge in terms of machine ressources and computation time for some of these approaches. It is surprising to be able to reach a decent result without having even used the movies features like genres.