Movielens Project

Andrew Blackford

7/24/2021

Movielens Project

Introduction The goal of the Movielens project is to create a recommendation system for movies using the movielens data set. The movielens data set has over 1 million ratings. The edx data set we will be working with contains 9000055 ratings from 69878 unique users who rated 10677 movies. These ratings will be the main component of creating the model to predict future ratings for the movie lens recommendation system. The movies were rated on a scale from .5-5 and the average rating is 3.5.
The data set is divided into a train and test set. The train set will be used to create a model that is compared to the actual ratings in the test set. The efficiency of the model is measured using RMSE function, looking for a score of 0.86490 or lower.
The first step to creating the model is to create a baseline prediction using the training set by taking the average rating of all the ratings and comparing it to the test set. Factoring in the user id and movie id will further lower the RMSE. Regularization using Lambda can be used to train the model to the desired RMSE score.

This code will create the data set for us.

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## Warning: package 'tidyverse' was built under R version 4.0.5
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.0     v forcats 0.5.1
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

```
## Warning: package 'tibble' was built under R version 4.0.5
```

```
## Warning: package 'tidyr' was built under R version 4.0.5
```

```
## Warning: package 'readr' was built under R version 4.0.5
```

```
## Warning: package 'dplyr' was built under R version 4.0.5
```

```
## Warning: package 'forcats' was built under R version 4.0.5

## -- Conflicts --------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret

## Warning: package 'caret' was built under R version 4.0.5

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table

## Warning: package 'data.table' was built under R version 4.0.5

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(tidyverse)
library(caret)
library(data.table)

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

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

ratings <- fread(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)
colnames(movies) <- c("movieId", "title", "genres")


# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


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

# 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
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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)

# create test sets

test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1,
                                  list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")


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

Methods/analysis

In order to create a model for predicting ratings we will need to briefly analyze the dataset. By looking at the dataset we can see it has 6 columns, the most important being user id, movie id and rating. The model will use the rating parameter to create the predictor. UserId and movieId will be the main categories we will use when fine tuning the model. The other two columns that might prove useful are genre and year/title. The movie id encompasses the year/title, making this category less valid. The genre parameter was not necessary for the final model.

This gives us a basic overview of the data.

```
head(edx$rating)
```

```
## [1] 5 5 5 5 5 5
```

userId movieId rating timestamp 1: 1 122 5 838985046 2: 1 185 5 838983525 3: 1 292 5 838983421 4: 1 316 5 838983392 5: 1 329 5 838983392 6: 1 355 5 838984474 title 1: Boomerang (1992) 2: Net, The (1995) 3: Outbreak (1995) 4: Stargate (1994) 5: Star Trek: Generations (1994) 6: Flintstones, The (1994) genres 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

Here we can look at the amount of ratings, and how many users and movies we have.

```
nrow(edx)
```

```
## [1] 9000061
```

```
n_distinct(edx$userId)
```

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

```
n_distinct(edx$movieId)
```

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

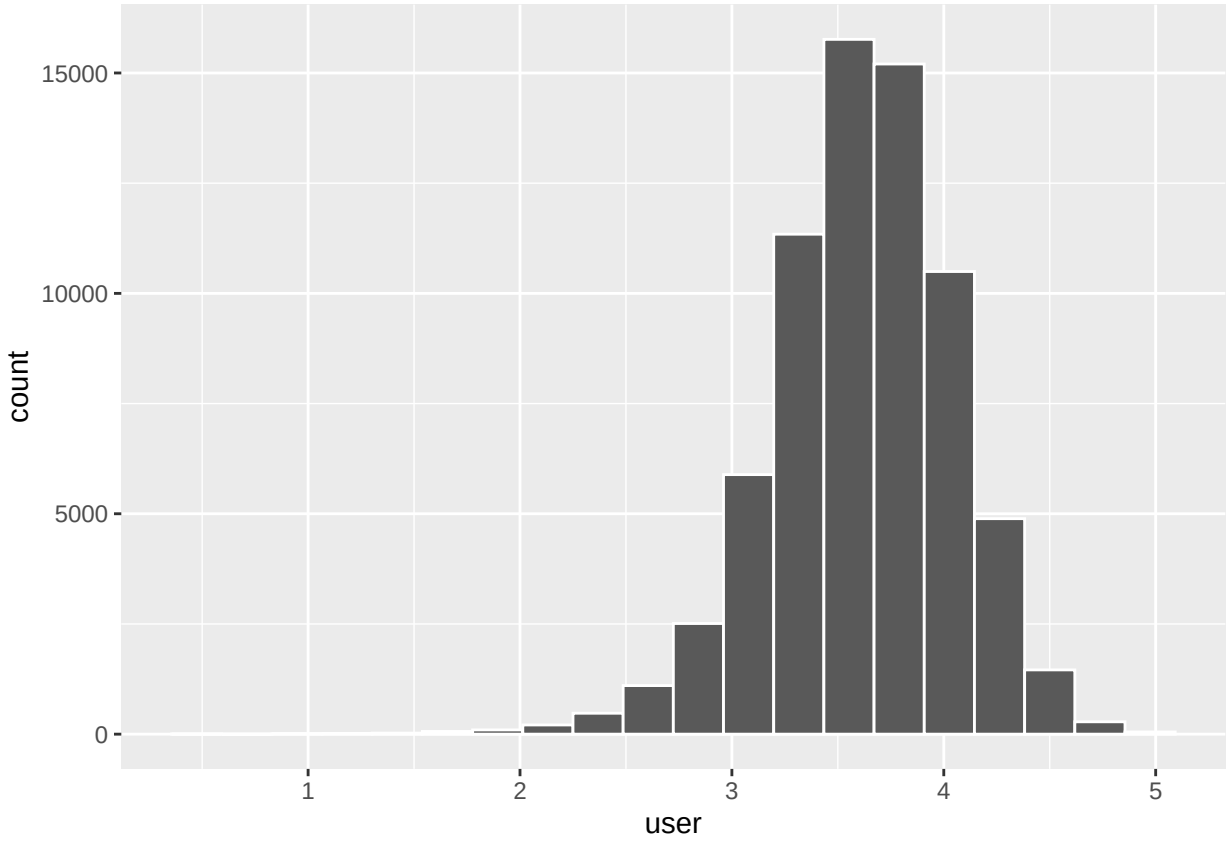
```
n_distinct(edx$genre)
```

```
## [1] 797
```

As the avg rating for users and movies will be a large part of the model we can create a graph that will give us a general idea of the ratings.

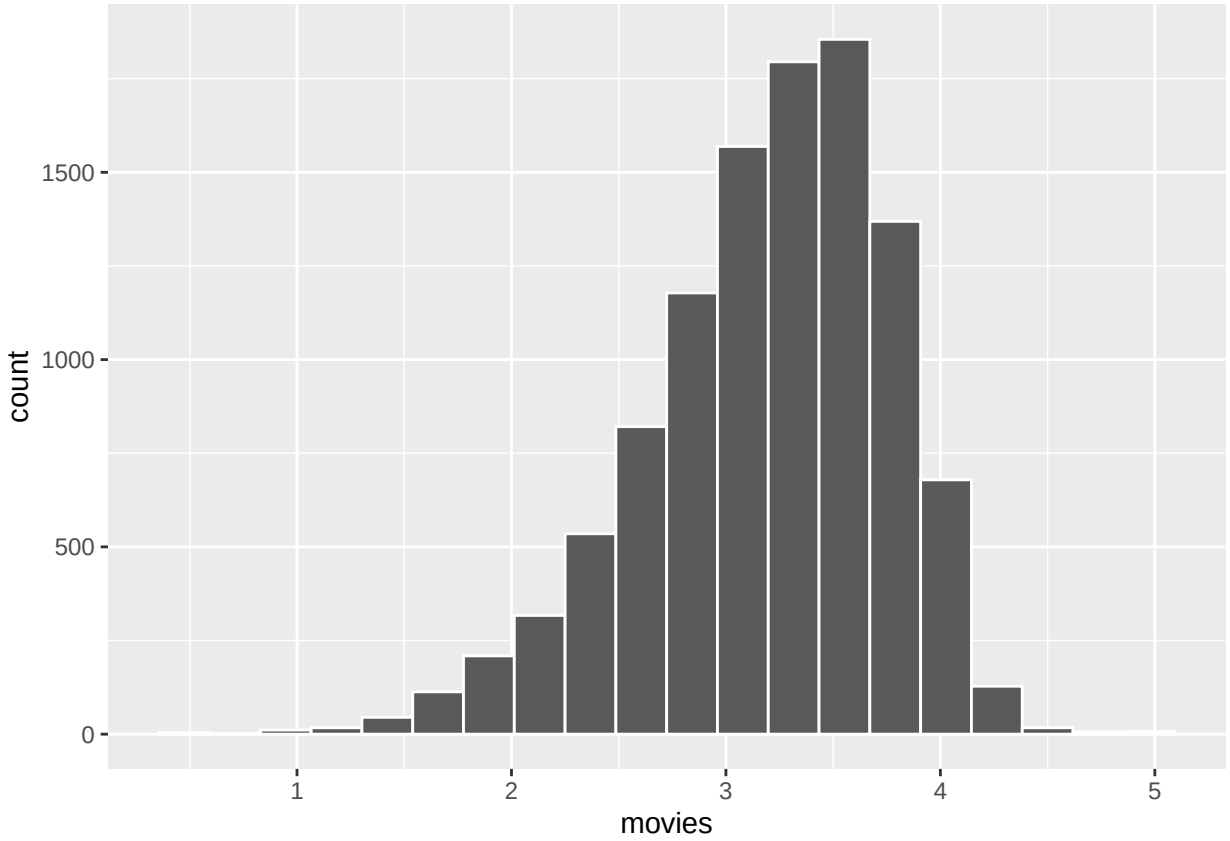
Here is the graph showing the user ratings.

```
edx %>%
  group_by(userId) %>%
  summarize(user = mean(rating)) %>%
  ggplot(aes(user)) +
  geom_histogram(bins = 20, color = "white")
```



Here is the graph showing the average movie rating.

```
edx %>%
     group_by(movieId) %>%
     summarize(movies = mean(rating)) %>%
    ggplot(aes(movies)) +
    geom_histogram(bins = 20, color = "white")
```



The first model we will use will be our baseline, which is the average rating of all the ratings and is stored as mu. I then used the RMSE function to compare the mu with the test set.

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

```
## [1] 3.512452
```

```
RMSE(baseline, test_set$rating)
```

```
## [1] 1.05975
```

The unique users and each individual movie's rating will be the most important to use when fine tuning the model. We will create a data set for the movie ratings and for the user ratings.

```
movie <- train_set %>%
  group_by(movieId) %>%
  summarize(movie = mean(rating - baseline))

predicted_ratings <- baseline + test_set %>%
  left_join(movie, by='movieId') %>%
  pull(movie)
RMSE(predicted_ratings, test_set$rating)
```

```
## [1] 0.9424221
```

We will do the same with the userid parameter.

```
user <- train_set %>%
  group_by(userId) %>%
  summarize(user = mean(rating - baseline))

predicted_ratings <- baseline + test_set %>%
  left_join(user, by='userId') %>%
  pull(user)
RMSE(predicted_ratings, test_set$rating)
```

```
## [1] 0.9777282
```

When we combine these two data sets into the predictor we get a better RMSE score.

```
user <- train_set %>%
  group_by(userId) %>%
  summarize(user = mean(rating - baseline))
movie <- train_set %>%
  group_by(movieId) %>%
  summarize(movie = mean(rating - baseline))

predictedRatings <- test_set %>%
  left_join(movie, by='movieId') %>%
  left_join(user, by='userId') %>%
  mutate(pred = baseline + movie + user) %>%
  pull(pred)
RMSE(predictedRatings, test_set$rating)
```

```
## [1] 0.8844204
```

Controlling for the user and movie has improved the RMSE. Regularization is a method used to create another parameter to modify the model. Lambda is a symbol used for the PLS method. With this method we will control for the outlier ratings, these outliers can cause a larger error when predicting ratings. In order to find the correct tuning parameter we will use cross validation. As we saw earlier in the graphs, the margins can be ignored. This sequence for the regularization training will get an RMSE in the acceptable range.

```
lambdas <- seq(3, 5, 1)

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

  baseline <- mean(train_set$rating)

  movie <- train_set %>%
    group_by(movieId) %>%
    summarize(movie = sum(rating - baseline)/(n()+l))

  user <- train_set %>%
    left_join(movie, by="movieId") %>%
    group_by(userId) %>%
    summarize(user = sum(rating - movie - baseline)/(n()+l))
```

```
  predictedRatings <-
    train_set %>%
    left_join(movie, by = "movieId") %>%
    left_join(user, by = "userId") %>%
    mutate(pred = baseline + movie + user) %>%
    pull(pred)

  return(RMSE(predictedRatings, train_set$rating))
})
```

The final step is to use the model on the edx and validation sets. The lambda with the smallest rmse is the one we will add to our model. We just add it to the formula and use our predictor to get our final RMSE.

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

```
## [1] 3
```

```
movie <- edx %>%
  group_by(movieId) %>%
  summarize(movie = sum(rating - baseline)/(n()+lambda))

user <- edx %>%
  left_join(movie, by="movieId") %>%
  group_by(userId) %>%
  summarize(user = sum(rating - movie - baseline)/(n()+lambda))

predictedRatings <- validation %>%
  left_join(movie, by = "movieId") %>%
  left_join(user, by = "userId") %>%
  mutate(pred = baseline + movie + user) %>%
  pull(pred)

RMSE(predictedRatings, validation$rating)
```

```
## [1] 0.8650725
```

RESULTS The final RMSE ended up being 0.8648177. This model used regularization and the PLS method and controlled for user and movies to reach an adequate RMSE.

```
RMSE(predictedRatings, validation$rating)
```

```
## [1] 0.8650725
```

Conclusion The goal of this report was to take the Movielens data set and create a model for predicting ratings. The report went over some data analysis isolating the user ratings and movie ratings to be the most useful data to work with. Overall this is a simple model, it would need to have other parameters like genre or year added to the model to make it more robust. Future work would be adding those sections on to the model and more fine tuning of the lambda.