MovieLens Project Report

Stephen Jagun

03/13/2021

Introduction

Machine learning is integral to a highly-technological modern business. It describes data about customer bases as a whole or personalizes an experience for a single user. The aim of machine learning is to process data into helpful information and naturally intuitive solutions. Effective machine learning algorithms have attracted a lot of attention in recent years: for example, in 2006, Netflix placed a seven-figure bounty on a verified improvemnent to their movie recommendation system.

We build off the Netflix challenge premise and, more specifically, predict movie ratings for users in a large dataset. We train a linear model to generate predicted movie ratings and calculate the Root Mean Square Error (RMSE) of the predicted ratings versus the actual ratings.

This report is composed of four parts: the introduction has presented the problem, the summary describes the dataset and develops preliminary inquiries, the methods section establishes the model and implements in with the accompanying .R file, and the conclusion section shares the results.

Summary

We use the MovieLens 10M dataset that consists of 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. This leads to wildly varying numbers of ratings for each movie, with the most rated movie being *Pulp Fiction* (1994) with over 31,000 ratings and over 100 titles with a single rating.

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

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following objects are masked from 'package:DescTools':
##
##     MAE, RMSE
```

```
## 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
```

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

```
## The following object is masked from 'package:DescTools':
##
##     %like%
```

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

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

```
library(dslabs)
library(tidyverse)
library(dplyr)
library(caret)

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

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
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)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
# Most rated films
edx %>% group_by(title) %>%
  summarize(n_ratings = n()) %>%
  arrange(desc(n_ratings))
```

```
## # A tibble: 10,676 x 2
##    title                                                     n_ratings
##    <chr>                                                         <int>
##  1 Pulp Fiction (1994)                                           31362
##  2 Forrest Gump (1994)                                           31079
##  3 Silence of the Lambs, The (1991)                              30382
##  4 Jurassic Park (1993)                                          29360
##  5 Shawshank Redemption, The (1994)                              28015
##  6 Braveheart (1995)                                             26212
##  7 Fugitive, The (1993)                                          25998
##  8 Terminator 2: Judgment Day (1991)                             25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)  25672
## 10 Apollo 13 (1995)                                              24284
## # ... with 10,666 more rows
```

```
# Number of movies rated once
edx %>% group_by(title) %>%
  summarize(n_ratings = n()) %>%
  filter(n_ratings==1) %>%
  count() %>% pull()
```

```
## [1] 126
```

Due to the size of the dataset, built-in machine learning algorithms using R packages such as *caret* would require too much resources for a laptop to run in a timely manner if at all. Therefore, we achieve our goal using a linear model developed in the methods section of this report. The RMSE function is used from the DescTools package as the measure of accuracy.

The dataset is split 90-10 on train and test sets respectively. This is completed in the first steps of the script. The training set (edx) has 9,000,055 entries with 6 columns. Similarly, the test set (validation) has 999,999 entries and 6 columns. The column information is shown below for the validation dataset.

```
glimpse(validation)
```

```
## Rows: 999,999
## Columns: 6
## $ userId    <int> 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ movieId   <dbl> 231, 480, 586, 151, 858, 1544, 590, 4995, 34, 432, 434, 8...
## $ rating    <dbl> 5.0, 5.0, 5.0, 3.0, 2.0, 3.0, 3.5, 4.5, 5.0, 3.0, 3.0, 3....
## $ timestamp <int> 838983392, 838983653, 838984068, 868246450, 868245645, 86...
## $ title     <chr> "Dumb & Dumber (1994)", "Jurassic Park (1993)", "Home Alo...
## $ genres    <chr> "Comedy", "Action|Adventure|Sci-Fi|Thriller", "Children|C...
```

Methods

The simplest model is to use the average across every user and every movie as all of our predicted ratings. This model follows the simple equation,

$$Y_{u,i} = \mu, \tag{1}$$

where $Y_{u,i}$ is the predicted rating of user u and movie i and μ is the average rating across all entries. This is computed as 3.512 (`mean(edx$rating)`).

```
mu <- mean(edx$rating)
RMSE(validation$rating, mu)
```

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

In order to improve our model, we add an independent error term $b_{u,i}$ that expresses rating differences for users and movies. We will add the user bias term later, but for now we add the movie bias term b_i. This term averages the rankings for any movie i because some are liked or hated more than others. The new model is:

$$Y_{u,i} = \mu + b_i. \tag{2}$$

```
# add movie bias term, b_i
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
# predict all unknown ratings with mu and b_i
predicted_ratings <- validation %>%
  left_join(b_i, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
# calculate RMSE of movie ranking effect
RMSE(validation$rating, predicted_ratings)
```

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

Now we introduce the user bias term b_u in order to further improve our model. This term minimizes the effect of extreme ratings made by users that love or hate every movie. Each user u is given a bias term that sways their predicted movies. Our updated model is:

$$Y_{u,i} = \mu + b_i + b_u. \tag{3}$$

```
# add user bias term, b_u
b_u <- edx %>%
  left_join(b_i, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
# predict new ratings with movie and user bias
predicted_ratings <- validation %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
# calculate RMSE of movie and user bias effect
RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.8653488
```

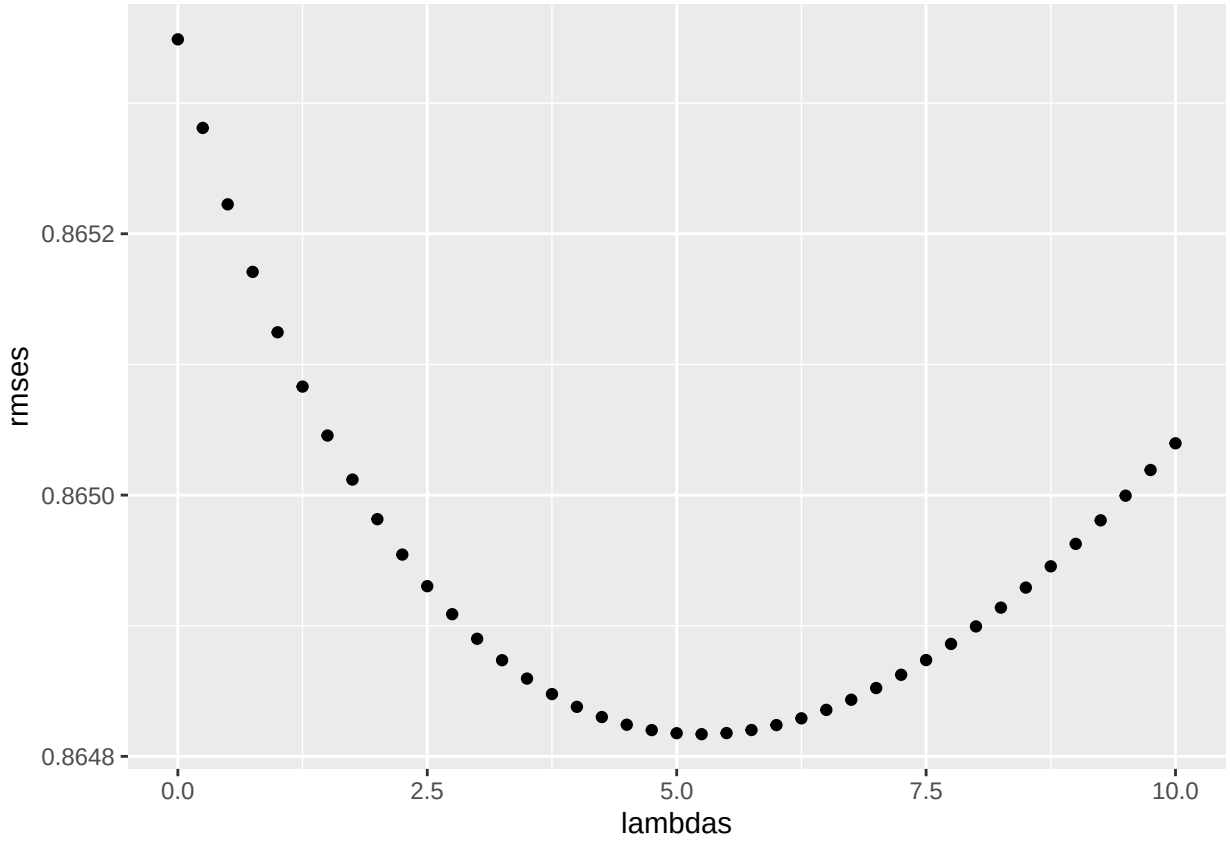
Finally, we employ regularization to reduce the effect of large errors in our predictions. Regularization penalizes incorrect estimates on small sample sizes. For instance, our b_i term accounts for the average deviation on all ratings of a movie, whether there is 1 or 100 ratings to the movie. We use regularization to reduce the dramatic effect that a exceptionally extreme rating will have on our b_i term. This method is also applied to the user bias term b_u to reduce large anomalies in the ratings of users.

Regularization achieves the same goal as confidence intervals when you are only able to predict a single number, not an interval. Our new model is:

$$\frac{1}{N}\sum_{u,i}(Y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2), \tag{4}$$

where the first term is our previous least squares equation and the last term is the penalty with large bias terms. Minimizing the biases using a single λ is the goal to our model shown above. We test `lamda <- seq(from=0, to=10, by=0.25)` and plot the results below:

```
qplot(lambdas, rmses)
```



We see that the minimizing λ term is

```
lambdas[which.min(rmses)]
```

```
## [1] 5.25
```

Results

For completeness, the final model is executed below:

```
# choose minimized lambda
lam <- lambdas[which.min(rmses)]
# compute regularize movie bias term
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lam))
# compute regularize user bias term
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lam))
# compute predictions on validation set based on these above terms
predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
```

```
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
# output RMSE of our final model
RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.864817
```

We can see incremental improvements to the RMSE as we supplant our model with bias terms and regularization.

Method	RMSE
Average	1.06120
Movie effect	0.94391
Movie and user effects	0.86535
Regularized movie and user effect	0.86481

Conclusion

Our model is significantly more efficient than machine learning algorithms from R packages applied to this large data set. Because of the simplicity of the linear model, we are able to predict movie ratings without cost to the resources at our disposal.