Movie Lens Project Report

Amber Palladino

July 29, 2019

Overview

This report describes my work on the MovieLens capstone project for the HarvardX online Data Science professional certificate program.

The project involves working with the MovieLens dataset (included with course materials). The dataset consists of ten million movie ratings, with a unique 'movieId' for each movie, and a unique 'userId' for each user who provided one or more movie reviews.

The goal of the project is to create an algorithm to predict how users will rate movies. To accomplish this, the ratings data was split into training and test sets. I used the training data to determine each movie's average rating across users who rated it, and each user's average rating across movies the user rated. I reduced the weight of averages from small groups of ratings, and used the results on the test data to predict how each user would rate movies.

Methods

The assignment includes code to download the MovieLens data from the internet as a text file, parse the text into a dataframe, convert it to "tidy" form (for analysis using the R tidyverse package), and divide it into training ('edx') and test ('validation') segments (90% and 10% of the data, respectively). The code ensures that any movies and users that are present in the validation data are also present in the training data.

```
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes
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")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")

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

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

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

I further divided the training dataset into a training set ('edx_train') and a test set ('edx_test') to be used for tuning (80% and 20%, respectively). As with the original training/validation split, all movies and users present in the test data are also present in the training data.

```r
# Create train and test sets within the training data for tuning
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = F)
edx_train <- edx[-test_index,]
edx_test_temp <- edx[test_index,]

# Make sure userId and movieId in edx_test set are also in edx_train set
edx_test <- edx_test_temp %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")

# Add rows removed from edx_test set back into edx_train set
removed <- anti_join(edx_test_temp, edx_test)
edx_train <- rbind(edx_train, removed)

# Remove temporary tables and variables
rm(test_index, edx_test_temp, removed)
```

I determined the average rating across all movies in the dataset ('mu'), and created a range of penalty values for evaluation ('lambdas'). The penalty values tested range from 0 to 10, incremented by .25 (for a total of 41 penalty values evaluated).

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

```
## [1] 3.51257
```

```r
# Generate a sequence of lambdas for tuning. Lambda will be used as a penalty, in order to
# reduce the weight of ratings from a small number of users
lambdas <- seq(0, 10, 0.25)
```

Using the training dataset I established for tuning, I compared each movie's average rating to 'mu', determining the movie's effect on the average rating. I applyed a penalty using each value of 'lambdas'. The penalty was used to regress the rating value toward the average in cases where few ratings existed.

Once the movie effect was determined, I determined how the user's average rating diveraged from the overall average of 'mu'. As with determining the movie effect, each value of 'lambdas' was applied.

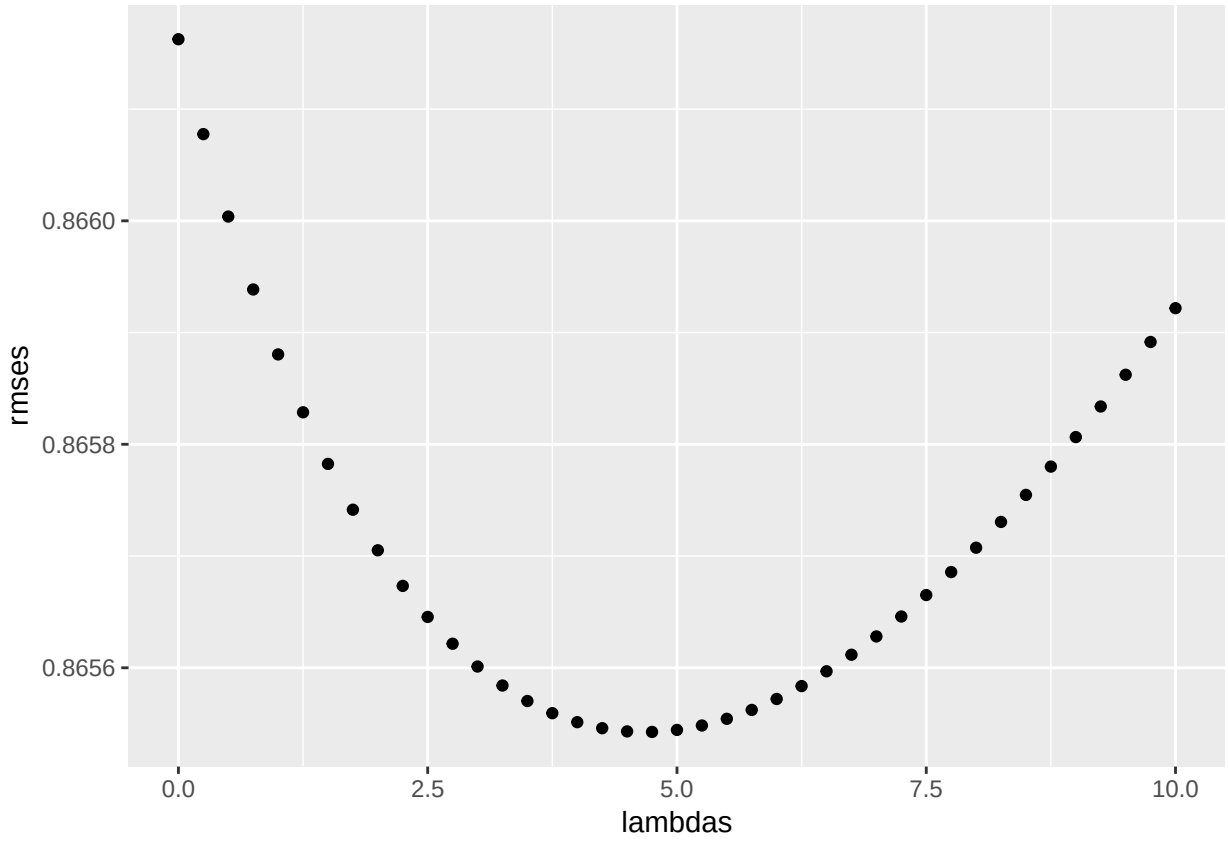
For each value of 'lambdas', the movie effects and user effects were joined to the test set created for tuning, and the algorithm generated ratings predictions for each movie/user combination. These predictions were then compared to the actual ratings within the test set. The success of each set of predictions was evaluated using RMSE (root mean squares error).

```
# Test various values of lambda
rmses <- sapply(lambdas, function(l){
  # Determine how each movie's average rating deviates from the average for all
  # movies (movie effect)
  b_i_tune <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i_tune = sum(rating - mu)/(n()+l))
  # Join the movie effect data back to the training data and determine user effect
  b_u_tune <- edx_train %>%
    left_join(b_i_tune, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_tune = sum(rating - b_i_tune - mu)/(n()+l))
  # Join movie effect and user effect data to the partitioned test data (from
  # the training data set)
  predicted_ratings_tune <- edx_test %>%
    left_join(b_i_tune, by = "movieId") %>%
    left_join(b_u_tune, by = "userId") %>%
    mutate(pred = mu + b_i_tune + b_u_tune) %>%
    .$pred

  # Evaluate RMSE of predictions and corresponding actual ratings from
  # the test data
  return(RMSE(predicted_ratings_tune, edx_test$rating))
})
```

The 'lambdas' value that produced the minimum RMSE value was identified and stored as 'best_lambda'.

```
# Identify lambda with smallest RMSE
qplot(lambdas, rmses)
```




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

```
## [1] 4.75
```

Using the 'best_lambda' penalty value, the full set of training data ('edx') was used to train the final algorithm, generating movie effect and user effect values for all records in the training data. Ratings predictions were then created for all movie/user combinations in the test dataset ('validation'). The predictions were compared to the actual ratings within the validation dataset, and a final RMSE value was calculated.

```
# Use the best lambda, movie effects, and user averages to generate predicted
# ratings for movies in validation data set

# Determine how each movie's average rating deviates from the average for all
# movies (movie effect)
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n() + best_lambda))
# Join the movie effect data back to the training data and determine user effect
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n() + best_lambda))
# Join movie effect and user effect data to the validation data
predicted_ratings <-
```

```
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

# Evaluate accuracy of predicted ratings for validation data against actual
# ratings for validation data, outputting RMSE
model_rmse <- RMSE(predicted_ratings, validation$rating)
model_rmse
```

```
## [1] 0.8648201
```

Results

The final RMSE value was 0.8648201, indicating that most predictions were within one star of the actual rating.

Conclusion

The ability to predict how a user will rate a movie he or she has not previously rated is a valuable means of creating customer satisfaction for a movie service provider. Movie ratings from other users and the user's ratings of other movies are useful inputs for generating predictions.

A weakness of this approach is in attempting to make predictions for a user who has not yet rated any movies, or has rated only a few movies. This weakness could potentially be mitigated by evaluating the effect of genre on ratings. The movie service provider might ask new users to indicate which genre(s) they prefer, and could then recommend highly rated movies within genre(s) that are of interest to the user.