HarvardX Data Science Project Report: Movie Recommendation Systems

Gretta Digbeu

May 30, 2019

A.Introduction

Netflix uses recommendation systems to predict how each of its users will rate *(i.e. how many stars it will assign to)* a particular movie. During the 2006 **Netflix Challenge**, the online streaming giant released movie rating data to the data science community with the promise of awarding $10 million to the team that could improve its movie recommendation algorithm by 10%. The winners were announced in 2009.

While the Netflix data is not publicly available, the **GroupLens** Research Lab at the University of Minnesota generated their own dataset of over 20 million ratings by more than 138,000 users for over 27,000 movies. This data, which was released to the learning community, is known as the the **MovieLens** dataset.

For the purposes of this exercise we use a subset of the **MovieLens** dataset that contains 10 million ratings by over 70,000 users for about 11,000 movies. Each row represents **one rating given by one user to one movie**, and has the following attributes: *user id, movie id, rating, time stamp, movie title*, and *movie genre*.

The purpose of the exercise is to generate a movie recommendation algorithm that predicts personalized ratings for the movies in our **Movielens** dataset. The resulting model then, suggests a unique rating for each movie a user has yet to rate. Because each outcome (rating of movie x by user y) has a different set of predictors, the machine learning problem at hand is unique.

B.Methods/Analysis

I. Data Prep

The code for extracting, cleaning, and partitioning the 10M **Movilens** dataset is provided in the course instructions. The resulting training and validation sets represent 90% and 10% of the data respectively, and contain the following columns: *user id, movie id, rating, time stamp, movie title, movie genre*.

II. Model Assessment Metric

Before generating a predictive model from the training data (edx), we define a function to compute the Root Mean Square Error (RMSE) of the resulting estimates. The RMSE is simply the square root of the Mean Squared Error (MSE), which is the most common metric used to capture the expected loss of an estimation model for continuous outcomes. For very large datasets such as **Movielens**, the MSE is equivalent to the expected value of the squared differences between our predictions and the actual values of our outcome. So the RMSE function looks like this:

```
RMSE <- function(true, predicted){sqrt(mean((true - predicted)^2))}
```

Our desired maximum RMSE is **0.87750**.

III. Model Parameters

We define the following parameters for our model:

- The average movie rating, *mu*;
- The regularized user-specific effects, *user effect*;
- The regularized movie-specific effects, *movie effect*.

These are all all derived from the **training set**, so that ultimately, for each combination of user i and movie j, the predicted rating is defined as:

`mu + user i effect + movie j effect`

In order to mitigate the negative impact that large errors can have on our final RMSE, we regularize the user effect and movie effect parameters. We do so by assigning a penalty term, *lamba*, which discounts the weight of estimates obtained from a small number of observations (i.e. very few users for a particular movie or very few movies for a particular user).

The movie and user effects are defined as follows:

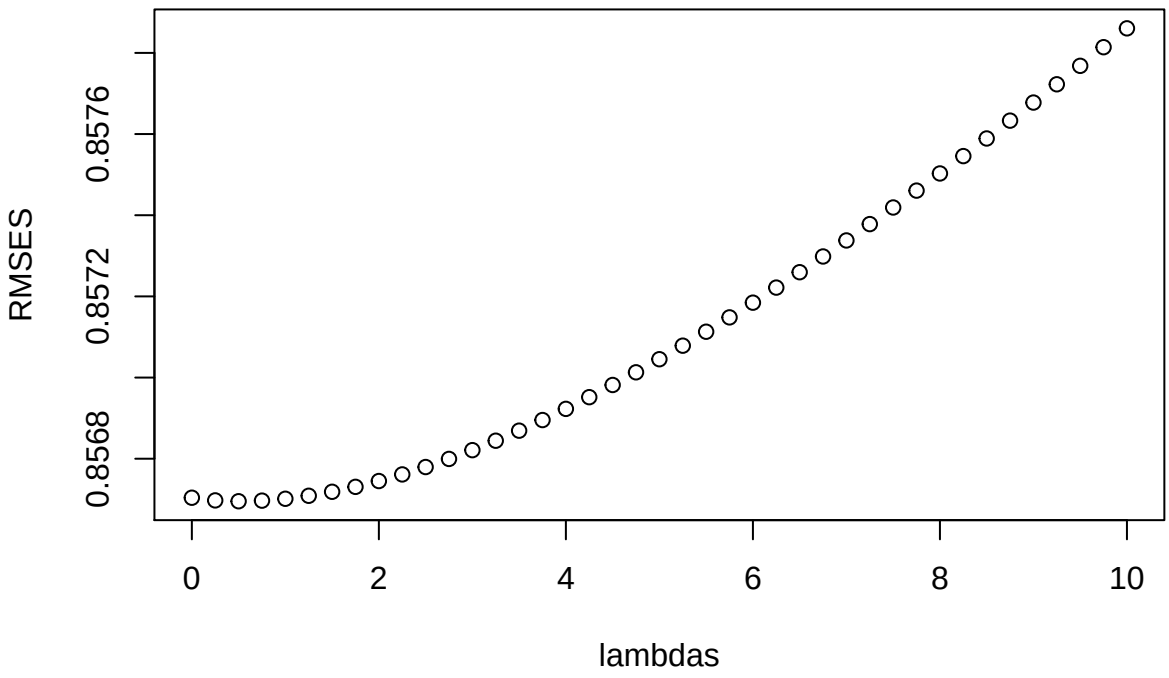
```
movie_effect <- edx %>%
  group_by(movieId) %>%
  summarize(f1 = sum(rating - mu)/(n()+lambda))

user_effect <- edx %>%
  left_join(movie_effect, by = "movieId") %>%
  group_by(userId) %>%
  summarize(f2 = sum(rating - mu - f1)/(n()+l))
```

IV. Model Optimization

Because *lambda* is a tuning parameter (different values can yield very different model estimates), we use cross validation across a range of values to optimize it. So the final algorithm looks like this:

```
lambdas <- seq(0, 10, .25)

RMSES <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)
  movie_effect <- edx %>%
    group_by(movieId) %>%
    summarize(f1 = sum(rating - mu)/(n()+l))

  user_effect <- edx %>%
    left_join(movie_effect, by = "movieId") %>%
    group_by(userId) %>%
    summarize(f2 = sum(rating - mu - f1)/(n()+l))

  predicted_ratings <- edx %>%
  left_join(movie_effect, by = "movieId") %>%
    left_join(user_effect, by = "userId") %>%
    mutate(prediction = mu + f1 + f2) %>%
    .$prediction

  return(RMSE(edx$rating, predicted_ratings))})
```

This code generates several Root Mean Squared Errors, each corresponding to a different value of *lambda*. After identifying the value of *lambda* that minimizes the RMSE (to 0.8566952), we are ready to test our model on the testing data.



V. Model Validation

Our final step thus consists of predicting user-specific ratings for the movies in the testing set, using the mean, regularized movie effect, and regularized user effect obtained from the training set (plugging in the optimized *lambda*).

The code for this final step looks like this:

```
mu <- mean(edx$rating)

final_movie_effect <- edx %>%
  group_by(movieId) %>%
  summarize(f1 = sum(rating - mu)/(n()+0.5))

final_user_effect <- edx %>%
  left_join(final_movie_effect, by = "movieId") %>%
  group_by(userId) %>%
  summarize(f2 = sum(rating - mu - f1)/(n()+0.5))

final_predictions <- validation %>%
  left_join(final_movie_effect, by = "movieId") %>%
  left_join(final_user_effect, by = "userId") %>%
  mutate(rating_hat = mu + f1 + f2) %>%
  .$rating_hat
```

Results

We see from the code above that the value of *lambda* that minimizes the RMSE on the training set is **0.5**. Once we plug this value into the final model and generate predictions for the testing data, we obtain an RMSE of **0.8652226**. This is below the desired cutoff of 0.87750.

Conclusion

In this exercise, we mimicked building a Netflix-like movie recommendation system that predicts how each user in a database will rate a particular movie. Using a subset of the popular **MovieLens** dataset which contains 10 million ratings by over 70,000 users for about 11,000 movies, we trained a simple model with regularized parameters using cross validation. Our final model yielded an RMSE of 0.8652226.