MovieLens Project Report

Aaron Kittel

4/20/2021

Introduction

Years ago, a challenge was extended by the Netflix video streaming service which offered a prize of $1,000,000 to whoever can provide the largest improvement to their movie recommendation system. The contest was won by BellKor's Pragmatic Chaos by improving Netflix's system by 10.06%.

In this exercise, we will perform a similar task by constructing our own recommendation system to predict what rating a user might give a movie they have not seen. We will establish a baseline metric and attempt to utilize machine learning algorithms in R to build a more effective model.

We will be utilizing a dataset which includes approximately ten million ratings given by Netflix users on a scale of 1 to 5. This includes the ratings themselves, a user ID, movie ID and title, a timestamp when the rating was given, and the movie genres.

For analysis, this data will be broken into three sets: a training set, a test set, and a validation set. We will use the training set to train various prediction models, the test set to check the viability of the models, and the validation set to estimate how a model might perform when employed against new real-world data. To compare our models, we will use the Root Mean Squared Error (RMSE).

Analysis

Naive Model As a baseline, we create a naive model where the predicted rating a user will give any movie is simply the average rating given by all users to all movies.

```
all_avgs <- mean(train_set$rating)
RMSE(test_set$rating, all_avgs)
```

```
##   model             rmse
## 1 Naive 1.05958580433716
```

We see that this naive model results in an RMSE of approximately 1.05959.

Adjusting for Biases We can further improve our predictions by calculating and incorporating averages for each individual user and movie into our model. This accounts for the biases users might have which result in them rating movies higher or lower than we would otherwise be expected, on average.

```
# Creating the movie averages for use as the movie bias
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - all_avgs))
```

```r
# Creating the user averages for use as the user bias
b_u <- train_set %>%
  left_join(b_i, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - all_avgs - b_i))

# Predicting ratings with user and movie biases included and adding RMSE to the data frame
biases_predictions <- test_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(pred = all_avgs + b_i + b_u) %>%
  pull(pred)

RMSE(test_set$rating, biases_predictions)
```

```
##         model             rmse
## 2 With Biases 0.86602407128937
```

The RMSE of this linear model, which now accounts for biases, improves drastically to approximately 0.86602.

Matrix Factorization To further reduce the RMSE, we might consider training a machine learning model to predict ratings. The most logical option would be to fit a K-Nearest Neighbors model to the data. This model would serve to make ratings predictions for a user based on a given number of similar users. However, we run into a problem with its implementation in that training such models on a dataset of this size is extremely computationally expensive and exceeds the capabilities of most personal computers. Unless we intend to purchase time on a mainframe, we need a more efficient model.

As an alternative, we have the option of performing Matrix Factorization using the "recosystem" library. This approach would turn our data into matrices with users as the rows, items (movies in our case) as the columns, and given ratings at the intersections. The model could then predict a user's unrated movies based on ratings given by other similar users. In this regard, Matrix Factorization operates very similarly to a K-Nearest Neighbors model in the way predictions are made.

To use Matrix Factorization in recosystem, we follow three simple steps:

1. Convert our training and test data into recosystem-compatible matrices

```r
train_reco <-  with(train_set, data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating = rating))
test_reco  <-  with(test_set,  data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating = rating))
```

2. Train a recosystem object on our training data matrix

```r
# Creating the recosystem object
fit_reco <-  recosystem::Reco()

# Fitting the recosystem object to the training data
fit_reco$train(train_reco)
```

3. Use the trained recosystem object to make predictions on our test data matrix

```
reco_predictions <- fit_reco$predict(test_reco, out_memory())
```

After completing these steps, we again check the RMSE of our predictions.

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

```
##                   model             rmse
## 3 Matrix Factorization 0.83390114425594
```

Our Matrix Factorization model achieves an RMSE of 0.83390. This level of improvement over our previous linear attempts suggests that the Matrix Factorization model is what we should employ against the validation dataset.

All three test set results may be found in the table below.

Method	RMSE
Naive Model	1.05959
With Biases	0.86602
Matrix Factorization	0.83390

Results

Having chosen the Matrix Factorization model to employ against the validation data with recosystem, we now simply need to train the model the combined training and test datasets to make predictions on the validation dataset. After doing so, we arrive at our final RMSE:

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

```
## [1] 0.8343784
```

Conclusion

We've achieved a respectably low RMSE utilizing a machine learning model efficient enough to be employed on large datasets using conventional personal computers. This is our desired outcome, however there remains room for further improvement.

The model we trained utilized raw rating data. We saw earlier in our analysis the significant improvements that may be made by accounting for biases in the data. It stands to reason that similar improvements can be made with similar adjustments done within the Matrix Factorization model.

Additionally, the recosystem model has parameters which may be optimized. For our purposes, the defaults were sufficient to achieve the desired reduction in RMSE. Exploring and optimizing these parameters would undoubtedly result in additional improvements.

Ultimately, our utilization of this basic Matrix Factorization model realized such significant improvements that we can confidently conclude it to be an ideal tool for addressing such problems in the future.