Movielens rating prediction

Miguel Ángel Jiménez Sánchez

May 3, 2019

1. Introduction

1.1. Objective

The objective of this project is creating a recommender system, that will predict the ratings that a set of users give to a set of movies. The system uses a set of parameters that will be fitted using data.

1.2. The dataset

On the GroupLens web page can be found several datasets publicly available regarding user ratings, based on the Movielens website.

The dataset used on this projects is the MovieLens 10M Dataset, that can be downloaded here. This data set, as the grouplens website states, has around 100,000 ratings from around 72000 users on around 10000 movies. The data is distributed in more than one file. The files come with no column names and the fields of each row are separated by '::'.

1.3. Main steps

The dataset is divided in two parts:

- Train set (edx): This is the data that will be used to fit the system.
- Validation set: This is a set of data that will be used to check the performance of the system on data that has not been used at all on the training process.

With these two sets, the main steps are the next:

1. Fit hiperparameters: A hiperparameter is a parameter whose value is used to control the learning process. In this model we have the parameter λ, used to penalize small samples (regularization). This step is divided in three parts:

a) Divide the train set (edx) into two sets. One to tune λ and the other a test set
b) Find the optimal λ using cross validation on the first set.
c) Check with the test set whether the model performs well with the λ selected.

2. Train the model with the hole train data frame (edx), using the tuned λ
3. Obtain the final RMSE with the validation data frame.

2. Methods/Analysis

2.1. Data preparation[1]

The data used on this system come from the Grouplens website in two files: movies.dat and ratings.dat. Each line on this file represents a register with the fields separated by '::'. The data comes with no column names, so they should be added. The information of both files is joined by movie id.

The resulting dataset has the next columns:

- userId: A number which identifies every single user
- movieId: A number which identifies every single movie
- rating: The rating given by each user to each movie. This is what the system will predict
- timestamp: indicates when the rating was given (in milliseconds since 1 January 1970)
- title: The title of the movie
- genres: The movie genres. A movie can have several genres.

The data frame obtained is divided in two parts. A 90% is used to train the model (this is called in the code as 'edx') and the other 10% is used to check the model performance (this is called in the code as 'validation')

2.2. Evaluation

The performance of the system will be measured based on the Residual Mean Squared Error (RMSE) applied to the validation set:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}\left(\hat{y}_{u,i} - y_{u,i}\right)^2}$$

Where:

- $y_{u,i}$: is the actual rating in validation given by user u to movie i
- $\hat{y}_{u,i}$: is the rating predicted by the system for user u and movie i
- N: is the number of user/movie combinations

2.3. Analysis

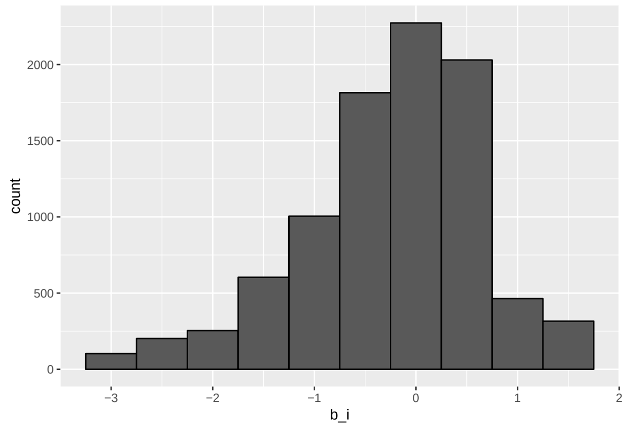
The three histograms on this section have been created from the movielens dataset on the dslabs package. This is a shorter version of the dataset that I've used to experiment my ideas. As a shorter version, the figures are an approximation of the dataset that will be used on this project.

2.3.1. Movie effect

As we know, some movies get better ratings than others. The next figure shows the distribution of movie average ratings minus the overall mean rating:

[1]The implementation of this part was provided by EDX.

```r
# movie effect
mu <- mean(movielens$rating)
movielens %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu)) %>%
  ggplot(aes(b_i)) +
  geom_histogram(bins = 10, color = "black")
```



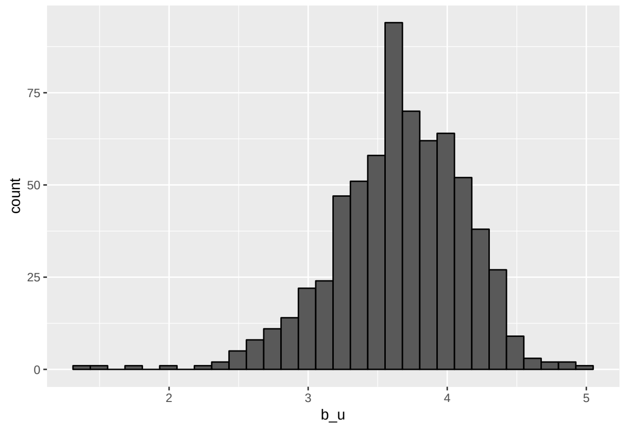
We can see a substantial variation.

2.3.2. User effect

Some users tend to be more generous than others with the ratings. The next figure shows the mean rating given by users who provided 100 or more ratings.

```r
# user effect
b_u <- movielens %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating))


b_u%>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```

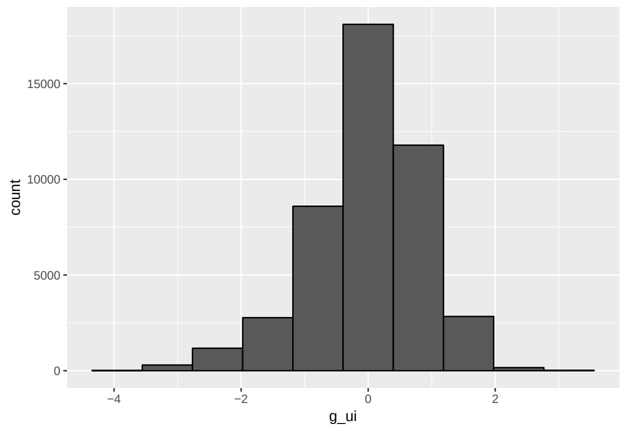


Here we see as well a substantial variation.

2.3.3. Genre preference effect

Some users prefer (or hate) more some genres than others. The next figure shows a distribution of the mean rating grouped by user and genres minus the user mean rating. He we consider each combination of genres as a category:

```r
# user preferences effect
movielens %>%
  left_join(b_u, by = "userId") %>%
  group_by(userId, genres) %>%
  summarize(g_ui = mean(rating-b_u)) %>%
  ggplot(aes(g_ui)) +
  geom_histogram(bins = 10, color = "black")
```



We see as well some variation.

2.4. Modeling approach

The system uses a model that consider the effect of three factors. First, some movies tend to have better ratings than others. Second, some users tend to give better rantings than others. And third, each user has genre preferences and give better ratings to those genres. A factor λ is included in order to penalize the effect of small sample sizes (which is called, regularization).

$$Y_{u,i} = \mu + b_i\left(\lambda\right) + b_u\left(\lambda\right) + g_{u,i}\left(\lambda\right) + \epsilon_{u,i}$$

Where:

- μ: is the mean rating in the train set
- b_i: is the movie effect, defined as the average rating for movie i minus the general mean rating μ.
- b_u: is the user effect, defined as the average rating given by user u minus the general mean and the movie effect.
- $g_{u,i}$: is the user genre preference effect, defined as the average rating given by the user to movies of the same genre of movie i minus μ, b_i and b_u. Each combination of genres is considered as a category (no genre separation).
- $\epsilon_{u,i}$: is the error in our model.

2.5. Regularization

The regularization parameter λ is used to penalize the average of a small size samples. For instance, consider how the b_i effect is calculated:

$$\hat{b}_i(\lambda) = \frac{\sum_{u=1}^{n_i}(Y_{u,i} - \hat{\mu})}{n_i + \lambda}$$

When there is a big n_i (sample size), then $n_i + \lambda \approx n_i$. But when n_i is small, then the estimate $\hat{b}_i$ is shrunken towards 0. The larger λ, the more estimate is shrunken.

So the parameter λ is what is called a hiperparameter that should be tuned. To do this we'll divide the training set in two parts. With the first part, we'll try several lambdas using cross validation. Then, the lambda with the lower RMSE will be tested with the second part of the original training set.

2.6. Cross validation

The cross validation consists on dividing the train data in n folds. For each fold, we'll train the model with the rest of the data and then calculate the RMSE on the current fold. The final RMSE is the mean of the RMSE of each fold.

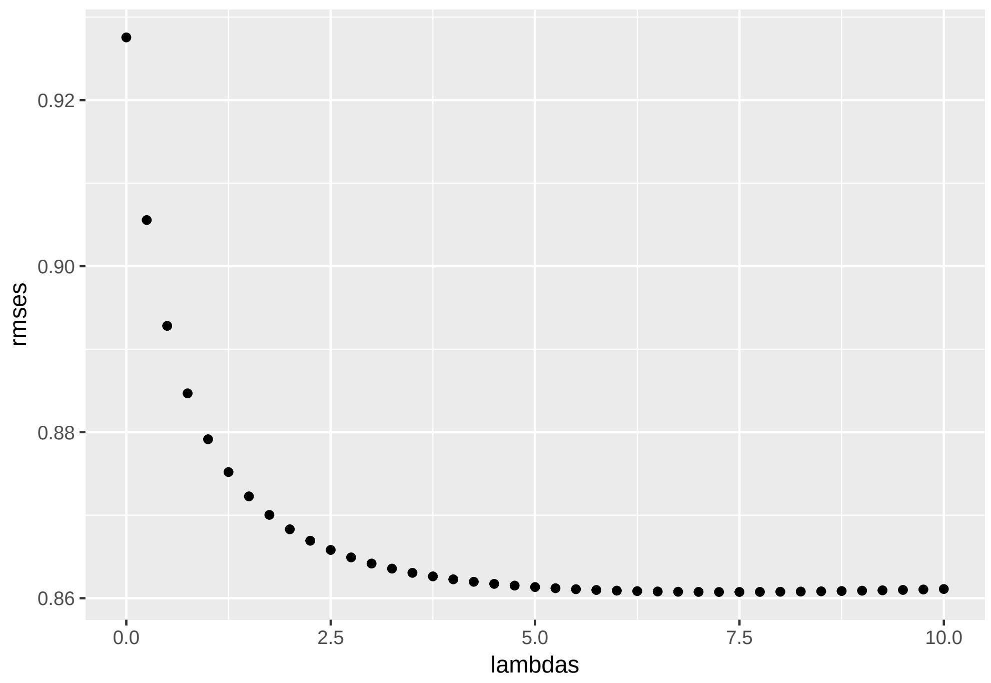
Calculating the estimators for each fold require several 'left_join' operations with large amounts of rows, which are expensive in execution time. To do these 'left_join' operations only once for each fold, first will calculate the different estimators with no penalization, but keeping the sample sizes (n_i for movies, n_u for users n_g for genres). Then for each lambda:

$$\hat{b}_i(\lambda) = \hat{b}_i(0) * \frac{n_i}{n_i + \lambda}$$
$$\hat{b}_u(\lambda) = \hat{b}_u(0) * \frac{n_u}{n_u + \lambda}$$
$$\hat{g}_{ui}(\lambda) = \hat{g}_{ui}(0) * \frac{n_g}{n_g + \lambda}$$

2.7. Implementation and results of lambda tuning

For the penalization parameter λ, we will examine the RMSE obtained for all the values from 0 to 10, in intervals of 0.25. To do this, we will divide the training set (edx) in two parts. We will call the first part the tuning set, which will be a 75% of the original training set (edx). The second part, a 25% of the original train set, will be the test set to validate the performance with the best lambda obtained with the tuning set.

The tuning set is divided in five folds. Then we will apply cross validation, as commented above, on our set of lambdas. The next figure shows the RMSE obtained for each λ.



The best lambda was 7.25. The figure seems to show a stabilization after this lambda, but actually the RMSE is increasing slightly. Applying this lambda on the test set, we get a RMSE of 0.8581286. This demonstrates that our model generalizes well on data not used in the training.

3. Results

3.1. Final result

After training the whole edx data set with the obtained lambda and calculating the RMSE on the validation set we obtain:

$$RMSE = 0.8554465$$

4. Conclusions

With this large amount of data, it was a challenge just to get the model to work on my old computer with only 4GB of RAM and obtain the final result in a reasonable time.

With the current model we still have margin for improvement. For example, for movies with several genres we can determine the effect of each individual genre. Or we can explore other approach with matrix factorization, using the 'recommenderlab' package. Those are possibilities that I would explore in the future.