Edx Capstone project - Movie Recommendation

Gregoire Mansio

12/05/2021

Executive summary

This document is the report of the creation of a movie recommendation algorithm developed in R, and is one of the two final projects for HarvardX Data Science Professional Certificate (PH 125x program).

It uses a comprehensive database of 10 million entries, gathering thousands of users and movies, and is based on previous works and examples from the Netflix challenge (see: https://en.wikipedia.org/wiki/Netflix_Prize)

The present report is organized as follows: I provide you first with the database presentation, then with the machine learning design, before finally moving to the outcomes of my work.

To obtain the final RMSE = 0.864827, I made use of five different variables, which are all quite basic in terms of intuition. These variables were analysed to create estimators associated to them. The main mathematical idea behind the computation of each estimator (and therefore behind the overall model) is the measurement of "deviations from average", and use them to guess a movie rating.

Analysis and model

Database and cleaning

The original database can be found here https://grouplens.org/datasets/movielens/. I am making use of the '10M' version. The full table is divided into 3 parts. The first and second parts are the 'train set' and 'test set' - and they form what I call the "Usable set" -. They are respectively gathering 7.2 and 1.8 million entries and are both extensively used to 'train' (find prediction patterns) and 'test' my algorithm (compare 'trained' predictions with observed 'test' data). The third part, the 'validation' set, is composed of the remainder of the data (1 million entries) and will be used only once when eventually measuring the precision of the final algorithm. It has the same role than the test set, but is used only at the very end of the project. Indeed, for evaluation purposes, such a set is required in order to put my algorithm to work in a "one-shot" application, with new data/information it hasn't encountered yet.

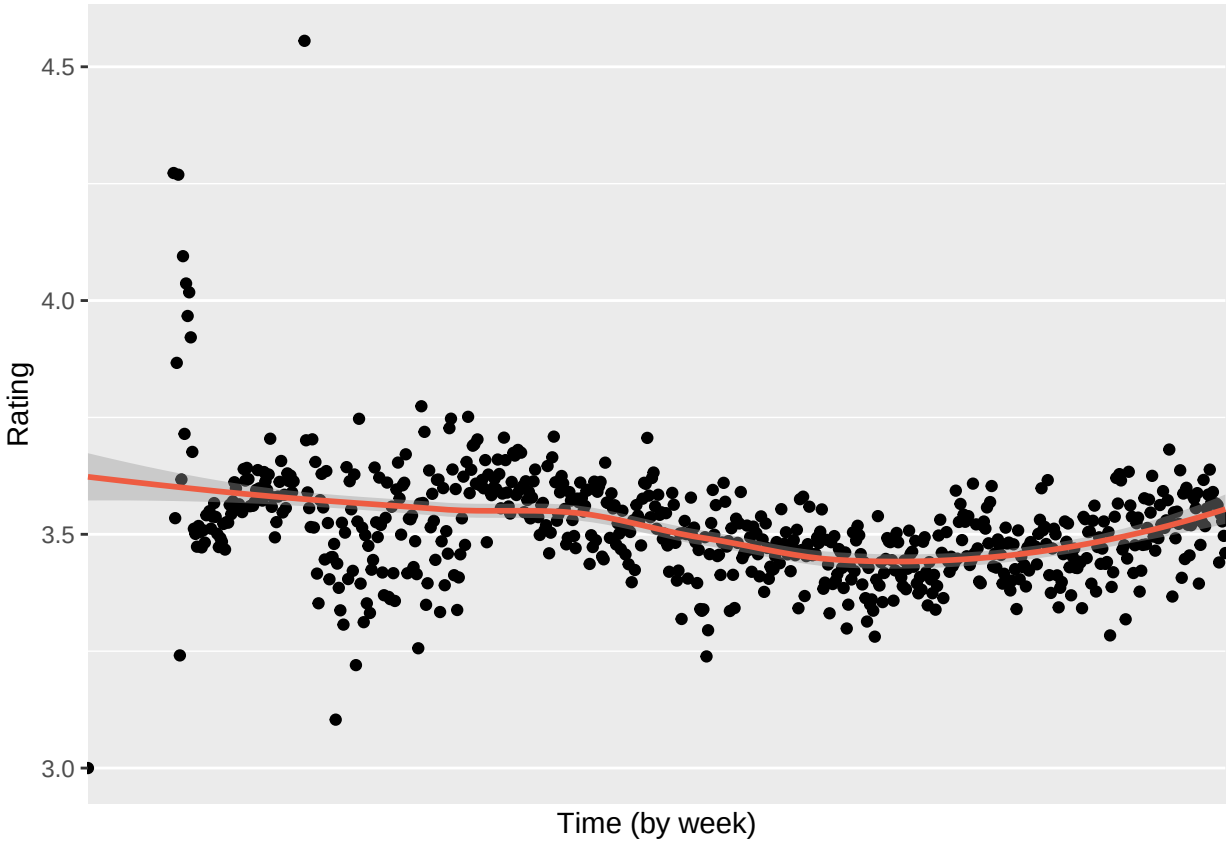
6 variables are stored on each row: The movie title, movieId (a unique Id for each movie), userId (a unique Id for each user), timestamp (the exact time when the rating was given), genres (the different genres of each movie), and the rating given.

The "Usable set" (7.2 + 1.8 = 9 Million) comprehends ratings given by 69878 users about 10677 movies. Ratings range from 0.5 to 5 stars, with an average of 3.51 stars. This average is the first and most basic estimator of the model.

The different cleaning operations aren't numerous, because the database has already been very properly created by the movielens group. Still, some work had to be done in view of being assured that every movie, user, movie age, and genre mixture (4 of my 5 variables) exist in both the train_set and test_set when comparing predictions and results. The same applies between the train_set and the validation_set. When doing this comparison, observations (movie, user, and age of movie) present in only one of the two sets are deleted.

Putting aside the first database creation steps, I want to underline that due to limited computing capacity (8Go RAM), I had to remove unused sets (like the validation set) and also unused variables, as well as

grouping as much as possible and deleting double information in order to be able to apply the techniques I wanted. For example, after having tried to include a date effect, the RMSE obtained was inconclusive, so I decided to delete the 'timestamp' variable from train and test sets to make things lighter. You can see in the following graph the effect of time (grouped by week) on the rating. The slight curvature of the smoothed line is not significant when computing the estimator.



For illustrative purposes, at the very end of the script, you can see other chunks of code referring to the operations on the date variable and on other abandoned strategies.

Estimators

The general idea is to build an algorithm step by step, each step providing a superior degree of precision to the model.

Because the dataset is so big, we are not able to use the caret package to make the software compute a linear regression with the standard lm() function. The idea is therefore to approximate the result of such a regression, by taking a given observation (a rating by user u to movie i), substracting the overall average (3.51), and then recursively substracting other elements (related to the movie type, the user's preferences etc) that can explain the difference between this particular rating and the other ones.

The first model is called a movie effect model, and only takes into account the average 3.51, and the average rating given to a particular movie by all users.

Movie effect I am regrouping observations as follows, and compute a regularized average. Note that this regularization process (inclusion of lambda in the calculation of the mean) is made to penalize estimators that were computed thanks to a relatively small number of observations. Here is the code used to produce the movie estimator 'bi_final'.

```
avg_mov_reg <- train_set %>%
  group_by(movieId) %>%
  mutate(bi_sum = sum(rating - avg), n_i=n()) %>%
  summarize(bi_final = bi_sum/(n_i+y))
```

'bi_final' captures the difference in the average rating of movie i compared to the overall average rating. Because there are 10641 different movies in the dataset, there are 10641 different bi_final. 'y' is the regularization term, which value's calculation will be explained later.

User effect The same applies for the user effect. I am computing the average rating given by a user to all the movies he has rated, and also use this as a potential effect in the observed rating.

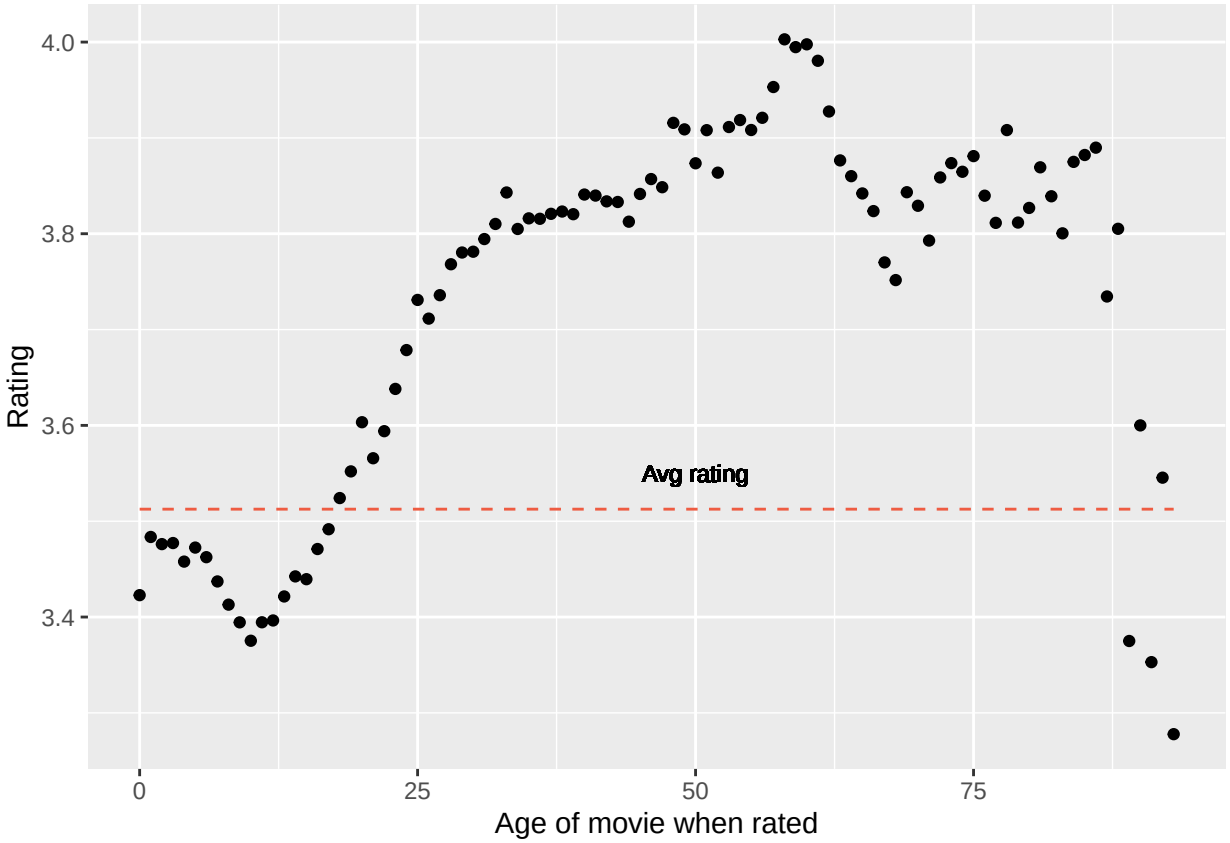
```
avg_usr_reg <- train_set %>%
  left_join(avg_mov_reg, by='movieId') %>%
  group_by(userId) %>%
  mutate(bu_sum = sum(rating - avg - bi_final), n_u=n()) %>%
  summarize(bu_final = bu_sum/(n_u+y))
```

Note something, I am recursively adding the desired effects, meaning that 'bi_final' is taken into account when computing 'bu_final', so I am sure the two effects don't overlap. Not substracting 'bi_final' would attribute all the difference (between rating and avg) to the user effect. In other words, you want to include 'bi_final' in the computation of the difference, so you are capturing a user effect on what is left after you have corrected for the movie effect.

Genre effect For the genre effect, I first separate every observation (i.e. every row) in several rows, each new one corresponding to a unique genre the movie has. For example, a movie having three genres (Action | Aventure | Thriller) in the initial 'genres' variable will now be spread on three lines, and will only comprehend a single genre in the genre variable. Then, by grouping by 'genres', I am able to individually isolate the effect of a given genre on average.

Another way of computing the 'genre' effect is simply to group_by() genres. This way, we do not capture the rating-power of each individual genre, but instead of each genre_mix. After doing so, we are left with 784 genre effects, representing 784 genres combinations. Note that this second way of taking into account the genre effect is less elegant but more effective in terms of final precision.

Age effect The release year is written down in each movie title. Combining this information (year of release) with the year of the rating, we are able to calculate the "age of rating", or in other words, how old a movie was when he was rated. Indeed, there is a clear impact of the age on the rating. We can see two opposite patterns. Young movies (less than 10 years old when rated) tend to have a rating lower than average. But after the 10y-old threshold, the older the movie, the better the rating (with the exception of very-old movies).



Regularization

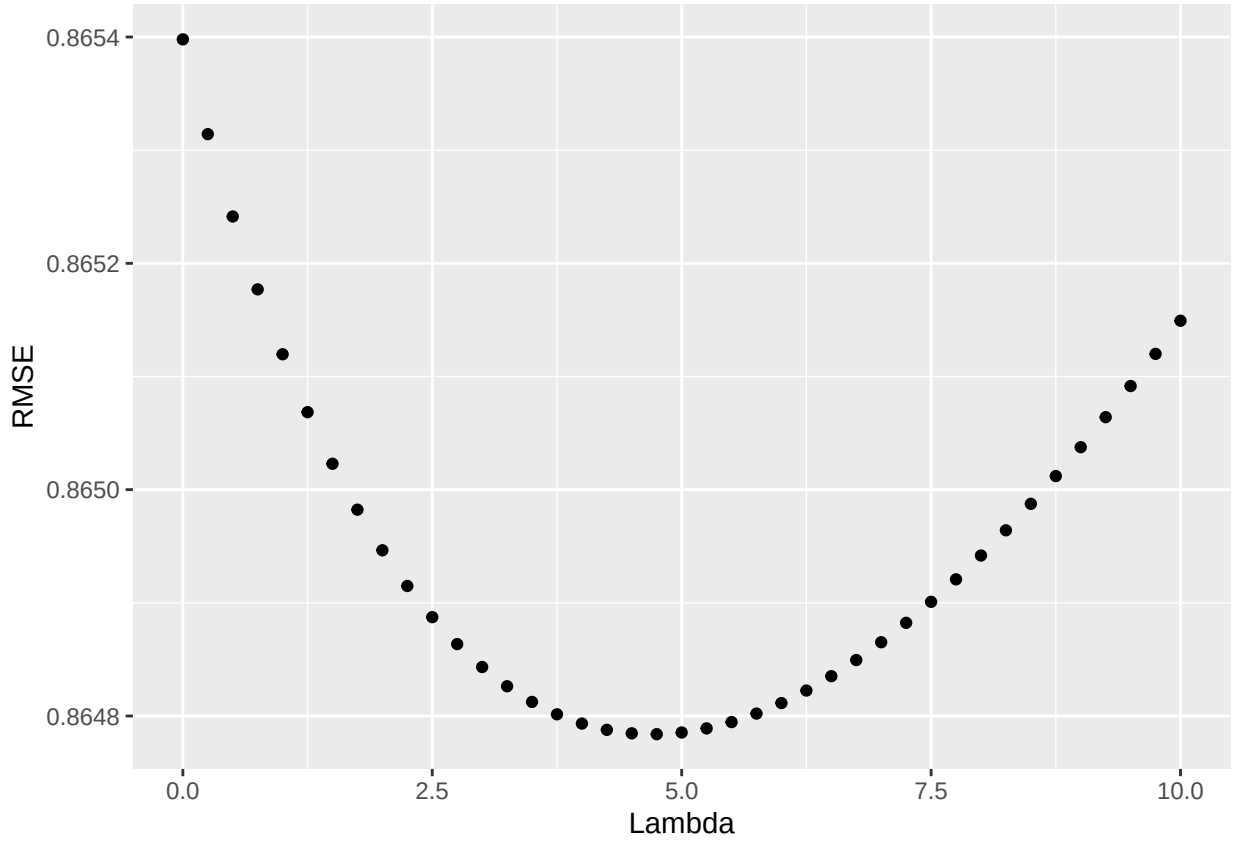
The idea of regularization is to penalize the weight of movies that have only been rated a very small number of times, as well as the rating given by users that have only given their feedback a few number of times. This technique is traditionally called 'lambda regularization', but for the sake of simplicity, we replaced 'lambda' by 'y'.

```
avg_mov_reg <- train_set %>%
  group_by(movieId) %>%
  mutate(bi_sum = sum(rating - avg), n_i=n()) %>%
  summarize(bi_final = bi_sum/(n_i+y))
```

In the above calculation of "bi_final", notice the 'y' term added to the calculation of the average. The 'y' penalizes movies that have been rated only a very few number of times (with small n_i), by synthetically modifying their count. The intuition behind is the unfairness that a movie averaging a 5 stars rating but only appearing three or four times in the database, has the same impact in the model than a movie rated 5 stars on average but appearing a hundred times. Same goes for users and genres.

As we aren't sure of the value 'y' should take, it is recommended to consider it a tuning parameter. This means we will try different values of this parameter, and keep the one leading to the best RMSE result. This operation is long, as the computer has to calculate as many models as number of y-values you want him to test. I computed the model for a sequence of y ranging from 0 to 10, with an increment of 0.25, leading to 41 possibles values for y, and therefore 41 RMSEs to estimate for each regularized model. As I regularized 3 models + the final model on the validation set, the computer calculated 164 models + the non-regularized one, making my script 1h08 to run on a modern laptop.

You can find below the plot of y-regularization (or lambda regularization) on the 'MUG²A reg' model. Note how the RMSE goes down around lambda = 4.75.



Final model

Following the creation of each estimator, I tested the same modeling design at almost every step. It is based on linear regression with a standard mathematical formula for this kind of technique.

$$Rating = avg + b_i + b_u + b_a + b_g + \epsilon$$

The idea is to add each estimator to the simple average, all of them computed thanks to the train_set, and then to compare this result (predicted "Rating") to the actual value observed in the test_set.

For illustrative purpose, here is the code used to perform this comparison.

```
validation_pred <- validation %>%
    left_join(avg_mov_reg, by='movieId') %>%
    left_join(avg_usr_reg, by='userId') %>%
    left_join(avg_genre_mix_reg, by='genres') %>%
    left_join(avg_age, by='age_rating') %>%
    mutate(pred = avg + bi_final + bu_final + bg_final + ba) %>%
    .$pred
  RMSE(validation_pred, validation$rating)
```

Results

I have computed a total of 8 models and their corresponding RMSEs. The table below summarizes the results of the different models, each row adding a superior degree of difficulty. Three interesting points are to bring forward. First, by looking at MU (Movie + User model) and MU reg (Movie + User regularized

model), we se a huge gap in precision, as the RMSE increases by almost 10% thanks to the regularization! Second, in terms of accuracy, it's even better to regularize the MU model than to add another estimator ('age'), as the non-regularized MUA model does not perform better than the MU reg. The same goes for the MUGA model (movie, user, age and genre) which also performs badly compared to the very simple MU reg. Third, about the genre effect. The elegant technique of trying to capture each individual genre effect and then reconstructing the prediction by adding those individual effects isn't very efficient. Indeed, the MUG²A genre_mix model is better performing than it's MUGA counter part.

method	RMSE
Basic average model	1.0607045
M - Movie effect model	0.9437144
MU - Movie + User model	0.8661625
MUA - Movie + User + Age model	0.8656926
MU reg - Movie + User regularized model	0.8655410
MUGA - Movie + User + Genre + Age model	0.8656096
MUG²A - Movie + User + Genre_Mix + Age model	0.8653595
MUG²A reg - Movie + User + Genre_Mix + Age regularized model	0.8647839

Finally, we see that this MUG²A regularized model has the best accuracy overall, and this is the one I decided to apply to the validation set, with the following RMSE.

```
## [1] 0.864827
```

Conclusion

With a RMSE of 0.864827, the final model can be considered quite good, as it obtains the best grading score possible for HarvardX project. However, some improvements could be made. For example, there is chances that a movie holding a unique age, and I did not regularized the 'age_rating' estimator. Moreover, after having finished the model as it is right now, it could have been possible to use the test_set as a supplement to the training_set, giving more information to the algorithm and hence more precision when testing it on the final Validation set.

It is also important to note that we only used a regression-like model to perform our predictions. There are multiple other techniques that can be employed, including matrix factorization, singular value decomposition (SVD) or principal component analysis (PCA). Those modeling strategies could perfectly present a better result than mine, but the time required to implement those alternatives is too large considering the deadlines.

Thank you for having read this report.