Movielens project

Predicting movie ratings by machine learning modeling

Contents

1 Introduction **2**

1.1 Overview . . . 2

1.2 Scope of the project . . . 2

2 Method of analysis **2**

2.1 Importing and checking movielens data set . . . 2

2.2 Choosing the parameters for modeling . . . 2

2.3 Additional parameters . . . 3

2.4 Creating training and test sets . . . 3

2.5 Deciding on modeling approach . . . 3

2.6 Naive model . . . 4

2.7 Regularized model . . . 4

2.8 Results on edX test set . . . 4

3 Results **4**

4 Conclusions **4**

5 Appendix **4**

5.1 Session info . . . 4

5.2 Full code . . . 4

1 Introduction

1.1 Overview

This report present a summary of **Movielens project**, which is part of Data Science Professional Certificate led by HarvardX on edX platform. The Movielens project was part of final graded assessments.

In the following sections some parts of code have been highlighted, but for reference full code has been included in the Appendix. This report, as well as a script file, can be found on GitHub repository.

1.2 Scope of the project

The scope of this project is to find a way to predict user ratings for a movie, based on chosen set of predictors from provided dataset, using possible machine learning techniques. The calculations will be based on GroupLens research lab Movielens 10M dataset, which consists of 10 million ratings on 10 thousand movies, made by 72 thousand users. Each rating has a different set of predictor. The goal of the project is to find a prediction method that would generate residual mean squared error (RMSE) lower than 0.86490.

2 Method of analysis

2.1 Importing and checking movielens data set

Starting the project I've had to import the Movielens data and create useful data set out of it. As a first step I download the zip file form web and transform the two dat files inside it (movies and ratings) using fread and str_split_fixed functions from data.table and stringr packages (part of tidyverse pack). Transformed data frames are saved to *movies* and *ratings* respectively. Both data frames are then joined together by *movieId* to *movielens* data frame that will be used onward. All code for those action was provided in the project description.

After creating *movielens* data set, I inspected it using glimpse function. Our movielens data set consist of 6 parameters:

- userId (integer)
- movieId (double class)
- rating (double class)
- timestamp (integer)
- title (character)
- genres (character)

2.2 Choosing the parameters for modeling

Establishing that *rating* is our outcome, I've considered the influence of the rest of the five parameters on the expected outcome. The PCA analysis couldn't be used to help with cumulative variance explanation, as *movielens* dataset is too big for machine calculations possibilities (and another transformation on character parameters would be needed).

The parameters that have been chosen as primary ones were *userId*, *movieId* and *genres*. *timestamp* and *title* parameters have been used to create additional parameters in different form (see next section).

2.3 Additional parameters

Additional two parameters were created and included in *movielens* data set, which we suspect might be influencing the rating gave by users.

First one is *year_of_release*, indicating in which year the movie has been released. It has been created by taking the years from *title* parameter, as those years are included at the end of character string in the brackets. It has been transformed into double-class using str_sub function from stringr package, and as.numeric function.

Another support parameter created to *movielens* data frame is *rateday*, which is the day of the week the rating has been done (assuming it's happening on the same day the movie is watched), in scale 1-7 where 1 is Monday. This parameter is included based on assumption that users rate differently if they are relaxed over the weekend, or it's middle of stressful week. *rateday* has been added to movielens data set as transformation of *timestamp* parameter by as_datetime and wday functions from lubridate package.

```
movielens <- mutate(movielens,
                    year_of_release = as.numeric(str_sub(title, start=-5, end=-2)),
                    rateday=wday(as_datetime(timestamp),week_start = 1))
```

2.4 Creating training and test sets

For the modeling purposes I've created a training (*edx*) and test (*validation*) sets out of movielens data frame. Test set *validation* is consisting of 10% of original *movielens* set. This part was done with createDataPartition function from the caret package. Code for creating *edx* and *validations* sets was provided as part of project description.

As test set *validation* has to be used only for final check of prepared prediction model, I've created additional test and training sub-sets of training set *edx*, with same approach as used before, thus creating *edx_train* and *edx_test* variables. The test set was created using 50% of *edx* data set. From now on, until final prediction model is created, all trainigs and tests would be done on those data frames.

In both cases I've made sure that *userId*, *movieId*, *year_of_release*, *rateday* and *genres* appearing in test sets are also in training sets. This has been done by using semi_join function on those two sets, by this selected parameters.

2.5 Deciding on modeling approach

Given the size of dataset, more complex algorithms for predicting *ratings* outcome, like random forest, K-nearest neighbors (KNN) or any regression forms couldn't be used (train function was taking too much time on given machine to calculate). I've decided to follow step by step Naive Bayes approach on all 5 chosen predictors. This could be described by following equation

$$\text{Y i,u,y,d,g} = \mu + \text{b i} + \text{b u} + \text{b y} + \text{b d} + \text{b g} + \epsilon \text{ i,u,y,d,g}$$

where:

- $Y_{i,u,y,d,g}$ - predicted movie rating
- μ - actual rating for all movies
- b_i - *movieId* effect
- b_u - *userId* effect
- b_y - *year_of_release* effect
- b_d - *rateday* effect
- b_g - *genres* effect
- $\epsilon_{i,u,y,d,g}$ - independent errors

2.6 Naive model

2.7 Regularized model

2.8 Results on edX test set

3 Results

0.864527388153627

4 Conclusions

5 Appendix

5.1 Session info

5.2 Full code