title: 'HarvardX: PH125.9x Data Science: MovieLens Rating Prediction Project' author: "Aditi Seshadri" date: "21/09/2020" output: html_document: default

pdf_document: default

Overview

This project is related to the Movielens Project as a part of the HarvardX:PH125.9x Data Science: Capstone Course. The report starts with a general idea of the project and by representing its objectives.

Then the given dataset will be prepared and setup. An exploratory analysis is carried out in order to develop a machine learning algorithm that could predict movie ratings until a final model. The results will be explained. Finally, the report ends with some concluding remarks.

Introduction

Dataset

```
###########################################################
# Create edx set, validation set (final hold-out test set)
###########################################################
# Note: this process could take a couple of minutes to load the tidyverse library

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(data.table)
library(dplyr)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl &lt;- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
```

```
ratings &lt;- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies &lt;- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) &lt;- c("movieId", "title", "genres")

# if using R 3.6 or earlier:
movies &lt;- as.data.frame(movies) %&gt;% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                            title = as.character(title),
                                            genres = as.character(genres))

movielens &lt;- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index &lt;- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx &lt;- movielens[-test_index,]
temp &lt;- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation &lt;- temp %&gt;%
  semi_join(edx, by = "movieId") %&gt;%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed &lt;- anti_join(temp, validation)
```

The algorithm is to be developed on the edx dataset, as the 'validation' dataset will be used to test the final algorithm.

Methods and Analysis

Data Analysis

We will display the first few rows of the edx subset, in order to become familiar with the dataset. There are 6 columns: 'userId','movieId','genre','rating','title' and 'timestamp'. Each row represents a single rating of a user for a given movie.

```
head(edx) %&gt;% print.data.frame()
```

A summary of the data shows that there is no missing values.

```
summary(edx)
```

The total number of unique movies and users in the edx dataset is about 69878 unique users and 10669 unique movies.

```
edx %&gt;%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

The code below shows that users have a preference to rate movies rather higher than lower as shown by the distribution of ratings below. 4 is the most common rating, followed by 3 and 5. 0.5 is the least common rating. In general, half rating are less common than whole star ratings.

```
edx %&gt;%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.25, color = "black") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +
  ggtitle("Rating distribution")
```

We can observe that some movies have been rated quite frequently , while some have very few ratings and sometimes, only one rating. This will be important for our model as very low rating numbers might result in untrustworthy estimate for our predictions. In fact 125 movies have been rated only once.

Therefore, regularisation and a penalty term will be applied to the models in this project. Regularizations are techniques used to reduce the error by fitting a function appropriately on the given training set and avoiding overfitting (the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably).

Regularization is a technique used for tuning the function by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function, so that the coefficients don’t take extreme values.

```
edx %&gt;%
  count(movieId) %&gt;%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of movies") +
  ggtitle("Number of ratings per movie")
```

```
edx %&gt;%
count(movieId) %&gt;%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "black") +
scale_x_log10() +
xlab("Number of ratings") +
  ylab("Number of movies") +
ggtitle("Number of ratings per movie")
```

As 20 movies that were rated only once appear to be obscure, predictions of future ratings for them will be difficult.

```
edx %&gt;%
  group_by(movieId) %&gt;%
  summarize(count = n()) %&gt;%
  filter(count == 1) %&gt;%
  left_join(edx, by = "movieId") %&gt;%
  group_by(title) %&gt;%
  summarize(rating = rating, n_rating = count) %&gt;%
  slice(1:20) %&gt;%
  knitr::kable()
```

We can observe that the majority of users have rated between 30 and 100 movies. So, a user penalty term need to be included later in our models.

```
edx %&gt;%
count(userId) %&gt;%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "black") +
scale_x_log10() +
xlab("Number of ratings") +
ylab("Number of users") +
ggtitle("Number of ratings given by users")
```

Furthermore, users differ vastly in how critical they are with their ratings. Some users tend to give much lower star ratings and some users tend to give higher star ratings than average. The visualization below includes only users that have rated at least 100 movies.

```
edx %&gt;%
  group_by(userId) %&gt;%
  filter(n() &gt;= 100) %&gt;%
  summarize(b_u = mean(rating)) %&gt;%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black") +
  xlab("Mean rating") +
  ylab("Number of users") +
  ggtitle("Mean movie ratings given by users") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  theme_light()
```

Modelling Approach

We write now the loss-function, previously anticipated, that compute the RMSE, defined as follows:

$$ RMSE = \sqrt{\frac{1}{N}\displaystyle\sum*{u,i} (\hat{y}*{u,i}-y_{u,i})^{2}} $$

with N being the number of user/movie combinations and the sum occurring over all these combinations. The RMSE is our measure of model accuracy. We can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If its result is larger than 1, it means that our typical error is larger than one star, which is not a good result. The written function to compute the RMSE for vectors of ratings and their corresponding predictions is:

```
RMSE &lt;- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

The lower the better, as said previously.

I. Average movie rating model

The first basic model predicts the same rating for all movies, so we compute the dataset's mean rating. The expected rating of the underlying data set is between 3 and 4. We start by building the simplest possible recommender system by predicting the same rating for all movies regardless of user who give it. A model based approach assumes the same rating for all movie with all differences explained by random variation : $$ Y*{u, i} = \mu + \epsilon*{u, i} $$ with $\epsilon*{u,i}$ independent error sample from the same distribution centered at 0 and $\mu$ the "true" rating for all movies. This very simple model makes the assumption that all differences in movie ratings are explained by random variation alone. We know that the estimate that minimize the RMSE is the least square estimate of $Y*{u,i}$, in this case, is the average of all ratings: The expected rating of the underlying data set is between 3 and 4.

```
mu &lt;- mean(edx$rating)
mu
```

If we predict all unknown ratings with $\mu$ or mu, we obtain the first naive RMSE:

```
naive_rmse &lt;- RMSE(validation$rating, mu)
naive_rmse
```

Here, we represent results table with the first RMSE:

```
rmse_results &lt;- data_frame(method = "Average movie rating model", RMSE = naive_rmse)
rmse_results %&gt;% knitr::kable()
```

This give us our baseline RMSE to compare with next modelling approaches.

In order to do better than simply predicting the average rating, we incorporate some of insights we gained during the exploratory data analysis.

II. Movie effect model

To improve above model we focus on the fact that, from experience, we know that some movies are just generally rated higher than others. Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. We compute the estimated deviation of each movies' mean rating from the total mean of all movies $\mu$. The resulting variable is called "b" (as bias) for each movie "i" $b*{i}$, that represents average ranking for movie $i$: $$Y*{u, i} = \mu +b*{i}+ \epsilon*{u, i}$$

The histogram is left skewed, implying that more movies have negative effects

```
movie_avgs &lt;- edx %&gt;%
  group_by(movieId) %&gt;%
  summarize(b_i = mean(rating - mu))
movie_avgs %&gt;% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"),
ylab = "Number of movies", main = "Number of movies with the computed b_i")
```

This is called the penalty term movie effect.

Our prediction improve once we predict using this model.

```
predicted_ratings &lt;- mu +  validation %&gt;%
  left_join(movie_avgs, by='movieId') %&gt;%
  pull(b_i)
model_1_rmse &lt;- RMSE(predicted_ratings, validation$rating)
rmse_results &lt;- bind_rows(rmse_results,
                          data_frame(method="Movie effect model",
                                     RMSE = model_1_rmse ))
rmse_results %&gt;% knitr::kable()
```

So we have predicted movie rating based on the fact that movies are rated differently by adding the computed $b*{i}$ to $\mu$. If an individual movie is on average rated worse that the average rating of all movies $\mu$ , we predict that it will rated lower that $\mu$ by $b*{i}$, the difference of the individual movie average from the total average.

We can see an improvement but this model does not consider the individual user rating effect.

III. Movie and user effect model

We compute the average rating for user $\mu$, for those that have rated over 100 movies, said penalty term user effect. In fact users affect the ratings positively or negatively.

```
user_avgs&lt;- edx %&gt;%
  left_join(movie_avgs, by='movieId') %&gt;%
  group_by(userId) %&gt;%
  filter(n() &gt;= 100) %&gt;%
  summarize(b_u = mean(rating - mu - b_i))
user_avgs%&gt;% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("black"))
```

There is substantial variability across users as well: some users are very cranky and other love every movie. This implies that further improvement to our model my be: $$Y*{u, i} = \mu + b*{i} + b*{u} + \epsilon*{u, i}$$ where $b*{u}$ is a user-specific effect. If a cranky user (negative $b*{u}$ rates a great movie (positive $b_{i}$), the effects counter each other and we may be able to correctly predict that this user gave this great movie a 3 rather than a 5.

We compute an approximation by computing $\mu$ and $b*{i}$, and estimating $b*{u}$, as the average of $$Y*{u, i} - \mu - b*{i}$$

```
user_avgs &lt;- edx %&gt;%
  left_join(movie_avgs, by='movieId') %&gt;%
  group_by(userId) %&gt;%
  summarize(b_u = mean(rating - mu - b_i))
```

We can now construct predictors and see RMSE improves:

```
predicted_ratings &lt;- validation%&gt;%
  left_join(movie_avgs, by='movieId') %&gt;%
  left_join(user_avgs, by='userId') %&gt;%
  mutate(pred = mu + b_i + b_u) %&gt;%
  pull(pred)
model_2_rmse &lt;- RMSE(predicted_ratings, validation$rating)
rmse_results &lt;- bind_rows(rmse_results,
                          data_frame(method="Movie and user effect model",
                                     RMSE = model_2_rmse))
rmse_results %&gt;% knitr::kable()
```

IV. Movie, user and genre effect model.

We will now analyze the effect of the genre on the movies and the users.

```
genre_avgs&lt;- edx %&gt;%
  left_join(movie_avgs, by='movieId') %&gt;%
  left_join(user_avgs, by='userId') %&gt;%
  group_by(genres) %&gt;%
  filter(n() &gt;= 100) %&gt;%
  summarize(b_g = mean(rating - mu - b_i-b_u))
genre_avgs%&gt;% qplot(b_g, geom ="histogram", bins = 30, data = ., color = I("black"))

genre_avgs &lt;- edx %&gt;%
  left_join(movie_avgs, by='movieId') %&gt;%
  left_join(user_avgs, by="userId") %&gt;%
  group_by(genres) %&gt;%
  summarise(b_g = mean(ratings-mu-b_i-b_u))

predicted_ratings_3 &lt;- validation%&gt;%
  left_join(movie_avgs, by='movieId') %&gt;%
  left_join(user_avgs, by='userId') %&gt;%
  left_join(genre_avgs, by='genres') %&gt;%
  mutate(pred = mu + b_i + b_u+b_g) %&gt;%
  pull(pred)
model_3_rmse &lt;- RMSE(predicted_ratings_3, validation$rating)
rmse_results &lt;- bind_rows(rmse_results,
                          data_frame(method="Genre, Movie and User effect model",
                                     RMSE = model_3_rmse ))
rmse_results %&gt;% knitr::kable()
```

Our rating predictions further reduced the RMSE. But we made stil mistakes on our first model (using only movies). The supposes “best “ and “worst “movie were rated by few users, in most cases just one user. These movies were mostly obscure ones. This is because with a few users, we have more uncertainty. Therefore larger estimates of $b_{i}$, negative or positive, are more likely. Large errors can increase our RMSE.

Until now, we computed standard error and constructed confidence intervals to account for different levels of uncertainty. However, when making predictions, we need one number, one prediction, not an interval. For this we introduce the concept of regularization, that permits to penalize large estimates that come from small sample sizes. The general idea is to add a penalty for large values of $b*{i}$ to the sum of squares equation that we minimize. So having many large $b*{i}$, make it harder to minimize. Regularization is a method used to reduce the effect of overfitting.

V. Regularized movie,user and genre effect model

So estimates of $b*{i}$ and $b*{u}$ are caused by movies with very few ratings and in some users that only rated a very small number of movies. Hence this can strongly influence the prediction. The use of the regularization permits to penalize these aspects. We should find the value of lambda (that is a tuning parameter) that will minimize the RMSE. This shrinks the $b*{i}$ and $b*{u}$ in case of small number of ratings.

```
lambdas &lt;- seq(0, 10, 0.25)
rmses &lt;- sapply(lambdas, function(l){

  mu &lt;- mean(edx$rating)
  b_i &lt;- edx %&gt;%
    group_by(movieId) %&gt;%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u &lt;- edx %&gt;%
    left_join(b_i, by="movieId") %&gt;%
    group_by(userId) %&gt;%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  b_g &lt;- edx %&gt;%
         left_join(b_i, by="movieId") %&gt;%
         left_join(b_u, by="userId") %&gt;%
         group_by(genres) %&gt;%
         summarize(b_g = sum(rating-mu-b_i-b_u)/(n()+1))

  predicted_ratings &lt;-
    validation %&gt;%
    left_join(b_i, by = "movieId") %&gt;%
    left_join(b_u, by = "userId") %&gt;%
    left_join(b_g, by = "genres") %&gt;%
    mutate(pred = mu + b_i + b_u) %&gt;%
    pull(pred)

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

We plot RMSE vs lambdas to select the optimal lambda

```
qplot(lambdas, rmses)
```

For the full model, the optimal lambda is:

```
  lambda &lt;- lambdas[which.min(rmses)]
lambda
```

For the full model, the optimal lambda is: 5.25

The new results will be:

```
rmse_results &lt;- bind_rows(rmse_results,
                          data_frame(method="Regularized movie and user effect model",
                                     RMSE = min(rmses)))
rmse_results %&gt;% knitr::kable()
```

\pagebreak

Results

The RMSE values of all the represented models are the following:

```
rmse_results %&gt;% knitr::kable()
```

We therefore found the lowest value of RMSE that is 0.8648170.

Discussion

So we can confirm that the final model for our project is the following:

$$Y*{u, i, g} = \mu + b*{i} + b*{u} + b*{g} + \epsilon_{u, i, g}$$

This model work well if the average user doesn't rate a particularly good/popular movie with a large positive $b_{i}$, by disliking a particular movie.

Conclusion

We can affirm to have built a machine learning algorithm to predict movie ratings with MovieLens dataset. The regularized model including the effect of user is characterized by the lower RMSE value and is hence the optimal model to use for the present project. The optimal model characterised by the lowest RMSE value (0.8648170) lower than the initial evaluation criteria (0.8775) given by the goal of the present project. We could also affirm that improvements in the RMSE could be achieved by adding other effect (genre, year, age,..). Other different machine learning models could also improve the results further, but hardware limitations, as the RAM, are a constraint.