Data Science: Capstone - MovieLens Recommendation Model Report

by Edin Ceman

14th February 2021

1. Introduction

The following report will give you an overview of my Data Science: Captone MovieLens project. In the Introduction section background information on the "MovieLens Collection" is provided. This is followed by a description of the used data set, the goal of the project and the key steps that were performed. In the Methods & Analysis section, the modeling approach, process and methods will be explained. The Results section will then discuss the results of the model. Finally, the Conclusion section will summarize the model outcome and discuss the limitations.

1.1 The MovieLens Collection

The MovieLens dataset is a publicly available source of information containing user ratings for movies over a defined period of time. This data was collected and made available by GroupLens Research on the web site (http://movielens.org). There are different kinds of sizes of the MovieLens data. While there is also a full-size version of the data set available, in this course we use the 10m version to facilitate the modeling process while also reducing the necessary computing power.

1.2 Dataset

The 10m dataset from MovieLens is a smaller version of the full-size dataset and contains 10,000,054 ratings and 95,580 tags applied to 10,681 movies by 71,567 users of the online movie recommender service MovieLens. From the dataset, the following information is extracted as part of the Data Science: Capstone course:

- **userId:** This is an integer representing a unique user ID. Movielens users were selected at random for inclusion. Their ids have been anonymized.
- **movieId:** This is a numeric representing a unique movie ID.
- **rating:** This is a numeric representing the user rating for a movie. It can display the following discrete values (0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0).
- **timestamp:** This is an integer representing the exact time when the rating for a certain movie by a certain user was made.
- **title:** This is a character value displaying the movie's full title.
- **genres:** This is a character representing the genre of the movie. Movies can be in multiple genres, in this case they are all listed and separated with "|".

1.3 Goal

The goal is to build a movie recommendation system based on the provided MovieLens data set. Following that, the aim of the recommendation system is to predict a rating for a certain user and a certain movie at a certain point in time based on his previous ratings and and behavior. To achieve this goal, our system captures different effects or biases in the data set to identify the parameters that shape a user rating. This information can then be used to provide movie recommendations to a certain user.

1.4 Key Steps

To build the movie recommendation system, the following key steps were performed:

- Data Initiation: First, the "10m MovieLens" data set is downloaded, and the relevant information is extracted, transformed and loaded into R. The downloaded data set is then split into an "edx" set (90%) for the purpose of developing, training and testing, and a final hold-out "validation" data set (10%) for the final evaluation of the recommendation system.
- Data Preparation: Next, the "edx" data set is checked for NAs, filtered and enriched with additional features for the model design. The "edx" set is then split again into a train (80%) and a test (20%) set to develop, train and evaluate the recommendation model.
- Model Design: In the next step, the recommendation model is build using the train set, and evaluated with the test set. The model is designed such that it captures certain biases or effects in the train data (e.g. user, movie, time and genre effect). Additionally, regularization is applied to the effects to account for overfitting. The Root Mean Squared Error (RMSE) is used to evaluate the model performance and to select the optimal and final model parameters.
- Model Evaluation: The final model is then evaluated using the final hold-out validation set. Predictions are calculated for the validation set and those are used in the RMSE to determine the final model performance.
- Results: The final model results are shown and discussed.
- Conclusion: Lastly, the report is summarized and limitations and potential future work is discussed.

2. Methods & Analysis

This section explains the approach for preparing the data and building the recommendation model. The data initiation and data preparation sections describe how the MovieLens data is loaded and prepared. On the basis of that, the section Model Design describes describes the rationale and the methods used to build the recommendation model. The Model Evaluation section describes the evaluation process and metrics used to determine the model performance.

2.1 Data Initiation

The relevant code to download and create the "edx" and "validation" set has been kindly provided in the edx course "Data Science: Capstone". The provided code downloads the "10m MovieLens" data from source, extracts and transforms the relevant information into R. The **movieId** is transformed into a numeric data type, the **title** and **genres** are transformed into a character data type. This data set is then split into a data set with the name "edx" and a data set with the name "validation". The "edx" set contains 90% (or 9,000,055 rows) of the 10m MovieLens data set and shall be used to build, train and optimize the recommendation model. The "validation" set contains 10% (or 999,999 rows) from the 10m data set and represents the final hold-out data set that is only used in the final evaluation of the recommendation model. It is important to mention that the validation set is **not used** for the development and training of the model.

```
## Loading required package: tidyverse

## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.6     v dplyr   1.0.4
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift

## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

2.2 Data Preparation

The edx dataset is used for the purpose of developing and training the recommendation model. Initially, the edx set is checked for NAs. The edx set does not have NAs, hence no filling of NAs has to be conducted. Next, the data set is enriched with additional information for the model development. The feature **date** is created from the **timestamp** since we want to include a "Time Effect" in the model later on. The assumption is that the time a rating Additionally, the edx set is filtered to include only users that have at least 15 movie ratings. The rationale is provided in the model development section. This step reduces the size of the edx set to 8,999,671 rows. Finally, the prepared edx data set **edxT** is then split again into a train set (80%) (or 7,199,736 rows) and a test set (20%) (or 1,799,889 rows) to be used in the model development and parameter optimization.

```r
#### My Capstone Project start from here ####
#### MovieLens Movie Recommendation Model ####

# Install additional packages if not available already
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: lubridate
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")

# Load additional libraries
library(lubridate)
library(ggplot2)

## Data Wrangling

#Check if we have NAs in the edx dataset:
sapply(edx, function(x) sum(is.na(x)))
```

```
##    userId   movieId    rating timestamp     title    genres
##         0         0         0         0         0         0
```

```r
# Create additional features and prepare edx data set.
# Assumption: The timestamp which indicates when a certain movie was rated by a certain user has an eff
# Convert timestamp to datetime and then to day to have a granular view.
edxT <- edx %>% mutate(date = round_date(as_datetime(timestamp), unit = "day"))
dim(edxT)
```

```
## [1] 9000055       7
```

```r
# Assumption: Users that rate more often have more experience and therefore a better judgment which wil
edxT <- edxT %>% group_by(userId) %>% filter(n() >= 15) %>% ungroup()
dim(edxT)
```

```
## [1] 8999671       7
```

```r
# Split edx data into train and test set
# Setting seed to 1 to make results reproducible with sample.kind = "Rounding" for R Version > 3.5.
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
# Create index to split data (80% train & 20% test)
testIndex <- createDataPartition(y = edxT$rating, times = 1, p = 0.2, list = FALSE)
trainSet <- edxT[-testIndex, ]
testSet <- edxT[testIndex, ]
```

```
## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
# Make sure UserId and MovieId are existing in train and test set by applying semi_join.
testSet <- testSet %>%
  semi_join(trainSet, by = "movieId") %>%
  semi_join(trainSet, by = "userId")
```

2.3 Model Development

The recommendation model we want to build is called **Regularized Recommendation Model with Movie, Restricted User, Time & Genre Effect**. The goal of the model is to predict a rating for a certain movie, by a certain user at a certain point in time using the MovieLens data. The model is designed such that it captures different biases or effects in the training data set to identify the **biases** or **effects** that influence the resulting rating of a user for a movie. For the model, we use a movie, user, time and genre bias. Those will be explained later in this section. For each of the selected effects, the concept of **regularization** is used to avoid overfitting. This issue arises when a model is too specific, i.e. such that it performs well on a well known data set, but rather poorly on an unknown data set.

To apply the concept regularization, we introduce a penalty parameter **lambda** to each bias. The penalty parameter is applied to penalize the effect of a bias such that it's influence on the predicted rating is adjusted. The penalty parameter can be considered as a tuning parameter. Hence, we will define a range of lambdas, and repeat the model training and evaluation against the test set to select the optimal lambda. From my trials, I narrowed down the optimal lambda range to be somewhere between 4.5 to 5.5 in 0.1 steps which equals 10 iterations to avoid excessive computing. We could also conduct a finer penalty parameter search, e.g. 1 to 10 in 0,001 which would result in 9,000 steps to optimize our algorithm. However this approach would most probably result in overfitting, hence we will stick to the selected sequence from 4.5 to 5.5 in 0.1 steps.

To build the recommendations model, we will use the train set and evaluate it with the test set to find our optimal lambda. We use the Root Mean Squared Error (RMSE) to evaluate the model performance. The RMSE is constructed such that it calculates the standard deviation of the residuals, i.e. the difference between the true rating and the predicted rating. The RMSE is always non-negative due to the residuals getting squared, and it therefore penalizes large deviations disproportionally. Hence it is sensitive to outliers. The smaller the RMSE is, the better is the performance of our model.

First, the average movie rating is calculated. Then, we want to capture certain effect as explained in the Data Science: Machine Learning course. The model contains the following effects:

- **Movie Effect:** This parameter captures the effect a certain movie has on the rating, while adjusting for the overall average rating mu and regularizing it.

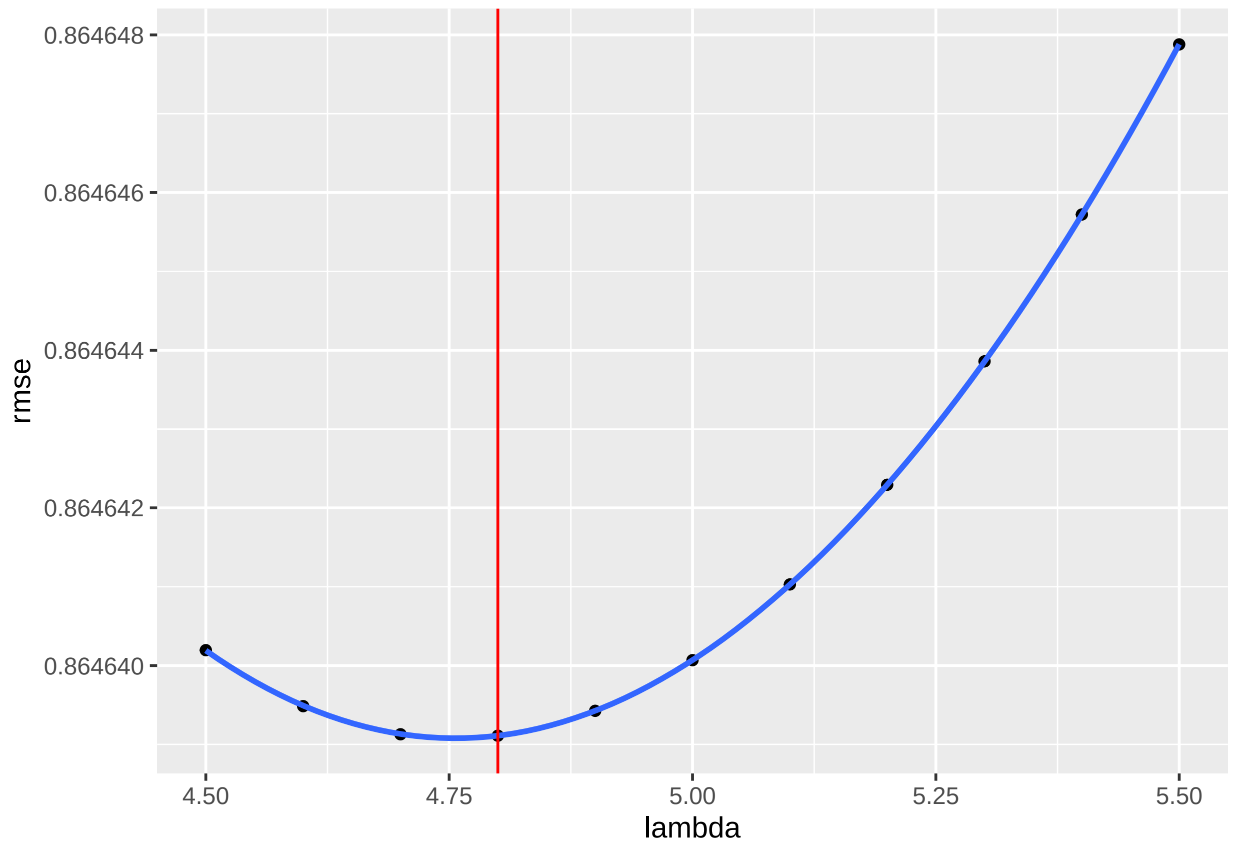
- **User Effect:** This parameter captures the effect a certain user has on the rating, while adjusting for the overall average and the movie effect and regularizing it.
- **Time Effect:** This parameter captures the effect the date (time) has when a rating was made has on the rating, while adjusting for the overall average mu and the movie and user effect and regularizing it.
- **Genre Effect:** This parameter captures the effect the genre of the movie has on the rating, while adjusting for the movie, user, and time effect ans regularizing it.

```r
lambdas <- seq(4.5, 5.5, 0.1)

# Training & evaluation of the model using the train & test set.
# The goal of the model is to capture the different biases or effects that have influence on a user rat
# The biases / effects are calculated based on selected features from the data set and determined in re
# The bias is then regularized to account for overfitting.
# The model is evaluated using the Root Mean Squared Error (RMSE) (with removal of NAs due to robustnes
RMSE <- function(true, predicted){
  sqrt(mean((true - predicted)^2, na.rm = TRUE))
}

# Training and Parameter Optimization
rmses <- sapply(lambdas, function(l){
  avg <- mean(trainSet$rating) # Feature Composition: Overall average rating: avg.
  movie_avg <- trainSet %>% # Feature Composition: Movie average for filling NAs later on.
    group_by(movieId) %>%
    summarize(movie_avg = mean(rating))
  b_movie <- trainSet %>% # Feature Composition: Regularized Movie bias: b_movie.
    group_by(movieId) %>%
    summarize(b_movie = sum(rating - avg)/(n()+l))
  b_user <- trainSet %>% # Feature Composition: Regularized User bias: b_user.
    left_join(b_movie, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_user = sum(rating - avg - b_movie)/(n()+l))
  b_time <- trainSet %>% # Feature Composition: Regularized Time bias: b_time.
    left_join(b_movie, by='movieId') %>%
    left_join(b_user, by='userId') %>%
    group_by(date) %>%
    summarize(b_time = sum(rating - avg - b_movie - b_user)/(n()+l))
  b_genre <- trainSet %>% # Feature Composition: Regularized Genre bias: b_genre.
    left_join(b_movie, by='movieId') %>%
    left_join(b_user, by='userId') %>%
    left_join(b_time, by='date') %>%
    group_by(genres) %>%
    summarize(b_genre = sum(rating - avg - b_movie - b_user - b_time)/(n()+l))
  predicted_ratings <- testSet %>% # Perform prediction based on feature composition.
    left_join(movie_avg, by = "movieId") %>% # Left join of the created biases and features on the test
    left_join(b_movie, by = "movieId") %>%
    left_join(b_user, by = "userId") %>%
    left_join(b_time, by = "date") %>%
    left_join(b_genre, by = "genres") %>%
    mutate(pred = avg + b_movie + b_user + b_time + b_genre) %>% # Perform prediction based on feature
    mutate(pred = ifelse(is.na(pred), movie_avg, pred)) %>% # 1. Replace NAs with movie average. Ration
    mutate(pred = ifelse(is.na(pred), avg, pred)) %>% # 2. Replace remaining NAs with overall average.
    mutate(pred = ifelse(pred < 0, 0.5, ifelse(pred > 5, 5, pred))) %>% # Computed predictions are nume
    .$pred
  return(RMSE(predicted_ratings, testSet$rating))
})
```

We run the model using the train set, and search for the optimal parameter lambda by evaluating the RMSE using the test set. The following figure shows the outcome for the search:

```r
# Plot the rmses and the lambdas
optResults <- data.frame(lambda = lambdas, rmse = rmses)
optResults %>% ggplot(aes(lambda, rmse)) +
  geom_point() +
  geom_smooth() +
  geom_vline(xintercept = lambdas[which.min(rmses)], color = "red")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```r
#qplot(lambdas, rmses)

# Find optimal penalty parameter lambda where the RMSE is minimized.
lambda <- lambdas[which.min(rmses)]

# Print optimal lambda
lambda
```

```
## [1] 4.8
```

We have identified the optimal penalty parameter to be 4.8. We can run the model one more time with the optimal parameter.

```
# Perform evaluation on test & train set with optimal lambda only.
rmse <- sapply(lambda, function(l){
  avg <- mean(trainSet$rating) # Feature Composition: Overall average rating: avg.
  movie_avg <- trainSet %>% # Feature Composition: Movie average for filling NAs later on.
    group_by(movieId) %>%
    summarize(movie_avg = mean(rating))
  b_movie <- trainSet %>% # Feature Composition: Regularized Movie bias: b_movie.
    group_by(movieId) %>%
    summarize(b_movie = sum(rating - avg)/(n()+l))
  b_user <- trainSet %>% # Feature Composition: Regularized User bias: b_user.
    left_join(b_movie, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_user = sum(rating - avg - b_movie)/(n()+l))
  b_time <- trainSet %>% # Feature Composition: Regularized Time bias: b_time.
    left_join(b_movie, by='movieId') %>%
    left_join(b_user, by='userId') %>%
    group_by(date) %>%
    summarize(b_time = sum(rating - avg - b_movie - b_user)/(n()+l))
  b_genre <- trainSet %>% # Feature Composition: Regularized Genre bias: b_genre.
    left_join(b_movie, by='movieId') %>%
    left_join(b_user, by='userId') %>%
    left_join(b_time, by='date') %>%
    group_by(genres) %>%
    summarize(b_genre = sum(rating - avg - b_movie - b_user - b_time)/(n()+l))
  predicted_ratings <- testSet %>% # Perform prediction based on feature composition.
    left_join(movie_avg, by = "movieId") %>% # Left join of the created biases and features to the test
    left_join(b_movie, by = "movieId") %>%
    left_join(b_user, by = "userId") %>%
    left_join(b_time, by = "date") %>%
    left_join(b_genre, by = "genres") %>%
    mutate(pred = avg + b_movie + b_user + b_time + b_genre) %>% # Perform prediction based on feature
    mutate(pred = ifelse(is.na(pred), movie_avg, pred)) %>% # 1. Replace NAs with movie average. Ration
    mutate(pred = ifelse(is.na(pred), avg, pred)) %>% # 2. Replace remaining NAs with overall average.
    mutate(pred = ifelse(pred < 0, 0.5, ifelse(pred > 5, 5, pred))) %>% # Computed predictions are nume
    .$pred
  return(RMSE(predicted_ratings, testSet$rating))
})

# Print RMSE from train & test set.
rmse
```

```
## [1] 0.8646391
```

2.3 Model Evaluation

3. Results

Results section

Present the modeling results and discusses the model performance. To evaluate the model results, we use the root mean squared error (RMSE). The RMSE is a measure that... When evaluating our model against the validation set using the RMSE, we achieve a value of 0.85.

4. Conclusion

Section that gives a brief summary of the report, its limitations and future work. The report at hand has given you the key insights of my Capstone Project. Initially, the MovieLens Project and data set has been explained. Following that, the key steps have been described. Afterwards, the modeling process as well as the approach for construcing the model has been explained and reasoning has been provided. Finally, the results have been evaluated and presented. The limitations of this project are that we have only used a restricted dataset (10m) instead of the full-scope dataset. Also, not all available fields and information of the MovieLens data set have been used such that certain effects could not be evaluated. Furthermore, the limitations of the computing power of a regular notebook or desktop pc did not allow for the application of more advanced modeling techniques, e.g. using ensemble methods with Random Forests, KNN, GamLoess and many more.

0 points: The report is either not uploaded or contains very minimal information AND/OR the report appears to violate the edX Honor Code. 10 points: Multiple required sections of the report are missing. 15 points: The methods/analysis or the results section of the report is missing or missing significant supporting details. Other sections of the report are present. 20 points: The introduction/overview or the conclusion section of the report is missing, not well-presented or not consistent with the content. 20 points: The report includes all required sections, but the report is significantly difficult to follow or missing supporting detail in multiple sections. 25 points: The report includes all required sections, but the report is difficult to follow or missing supporting detail in one section. 30 points: The report includes all required sections and is well-drafted and easy to follow, but with minor flaws in multiple sections. 35 points: The report includes all required sections and is easy to follow, but with minor flaws in one section. 40 points: The report includes all required sections, is easy to follow with good supporting detail throughout, and is insightful and innovative.