Movie Recommendation

Andres Antury

14/01/2022

1. Executive summary

The aim of this project is to create a movie recommendation system. The dataset used for the recommendation system is the Movielens dataset given by the Capstone Course at EDX which is sourced from Grouplens Lab.

In order to create the Machine Learning algorithm for the recommendation system, the Movielens data was split using the Caret package to create the training, test and validation sets. The evaluation metrics used to achieve the best model were derived using the loss function to calculate Root Mean Square Error (RMSE). *(Irizarri, R. Introduction to Data Science, p. 512).*

The models compared included a simple random variable model, models incorporating movie and user effects, regularisation and matrix factorisation.

A comparison table was created to view the different RMSE achieved by the models trained and tested, including the final validation set.

2. Analysis

For the analysis, the course handout included the Movielens dataset which was originally compiled by Grouplens Lab. The dataset provided originated from *(https://grouplens.org/datasets/movielens/10m/)*

The Movielens dataset was partitioned using the Caret package in order to create the train set and the validation set. The validation set was created as 10% of the Movielens data as follows:

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

The following code allows the userId and movieId in the validation set to also be in the edx set and to add the rows removed from validation set back into edx set:

```
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

In order to train and test the different models, the partitioned Movielens dataset (edx set) was in turn partitioned again to create a train set and test set. These sets were created in a similar way as the validation set but this time the test dataset was 10% of the edx data.

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_data <- edx[-test_index,]
temp <- edx[test_index,]

test_data <- temp %>%
  semi_join(train_data, by = "movieId") %>%
  semi_join(train_data, by = "userId")

removed <- anti_join(temp, test_data)
train_data <- rbind(train_data, removed)
```

The evaluation metrics used to achieve the best model were derived using the loss function to calculate Root Mean Square Error (RMSE). The following code was used to calculate the RMSE. Note: This code was provided by the course handout.

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

The modeling techniques used for this project were: random variable model, user and movie effects, regularisation (using penalised least squares) and matrix factorisation (using the Recosystem package).

2.1 Basic Model

This model calculates the average rating on the train set and evaluates the model on the test set:

```
mu_hat <- mean(train_data$rating)

model_1_rmse <- RMSE(test_data$rating, mu_hat)
```

2.2 Movie Effect Model

This model builds upon the average calculation and includes the movie effect:

```
mu_hat <- mean(train_data$rating)
movie_averages <- train_data %>%
  group_by(movieId) %>%
  summarise(b_hat_i = mean(rating - mu_hat))

predicted_ratings <- mu_hat + test_data %>%
  left_join(movie_averages, by="movieId") %>%
  pull(b_hat_i)

model_2_rmse <- RMSE(test_data$rating, predicted_ratings)
```

2.3 Movie and User Effect Model

This model builds upon the average calculation and includes the both the movie effect and the user effect:

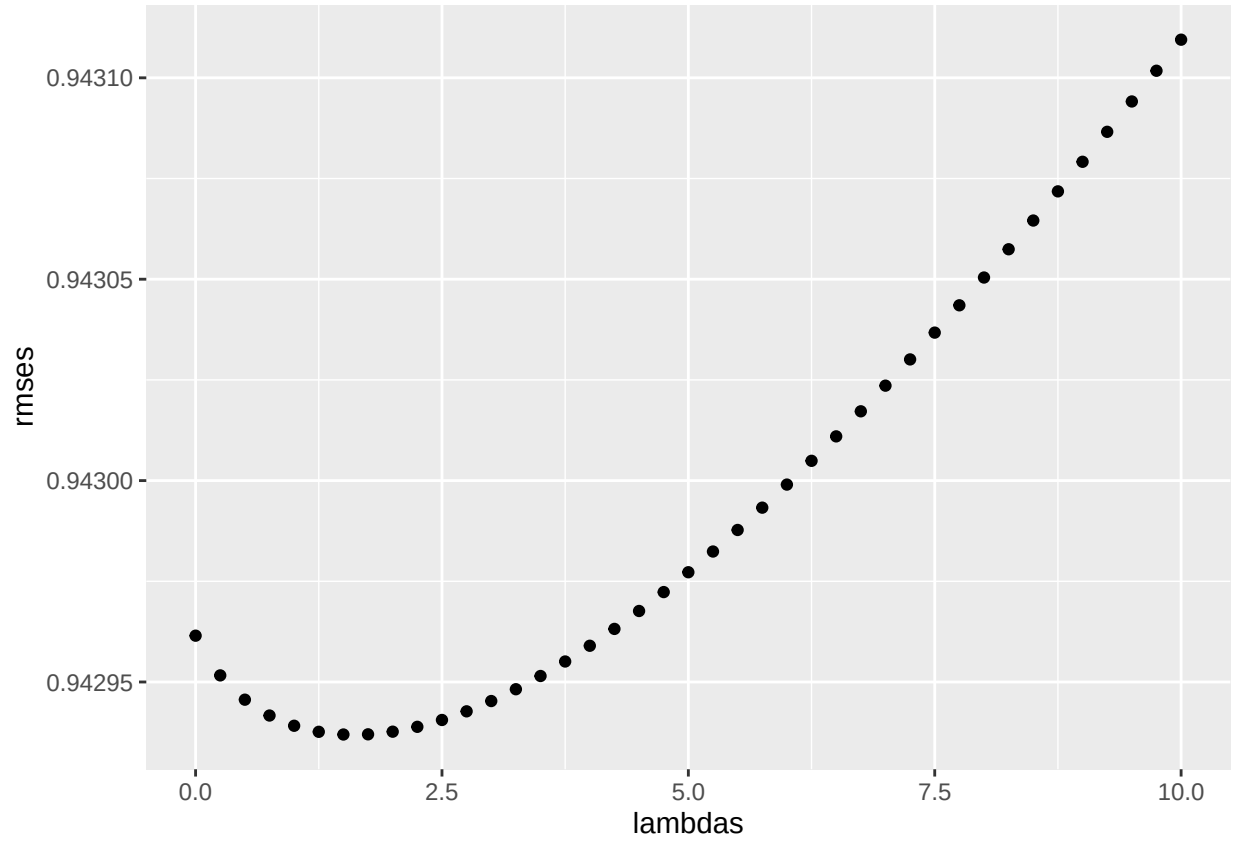
```
mu_hat <- mean(train_data$rating)
user_averages <- train_data %>%
  left_join(movie_averages , by="movieId") %>%
  group_by(userId) %>%
  summarise(b_hat_u = mean(rating - mu_hat- b_hat_i))

predicted_ratings <- test_data %>%
  left_join(movie_averages, by="movieId") %>%
  left_join(user_averages, by="userId") %>%
  mutate(pred = mu_hat + b_hat_i + b_hat_u) %>%
  pull(pred)

model_3_rmse <- RMSE(test_data$rating, predicted_ratings)
```

2.4 Regularised Movie Effect Model

The regularisation model uses penalised least squares and the tuning parameter lambda is calculated using cross validation *(Irizarri, R. Introduction to Data Science, p. 650)*. The lambda giving the lowest RMSE is used on the model as shown on the figure below.



The lambda value with the lowest RMSE is **1.5**

```
lambda <- 1.5 #This value is derived from the cross validation.

mu_hat <- mean(train_data$rating)
movie_regularised_averages <- train_data %>%
```

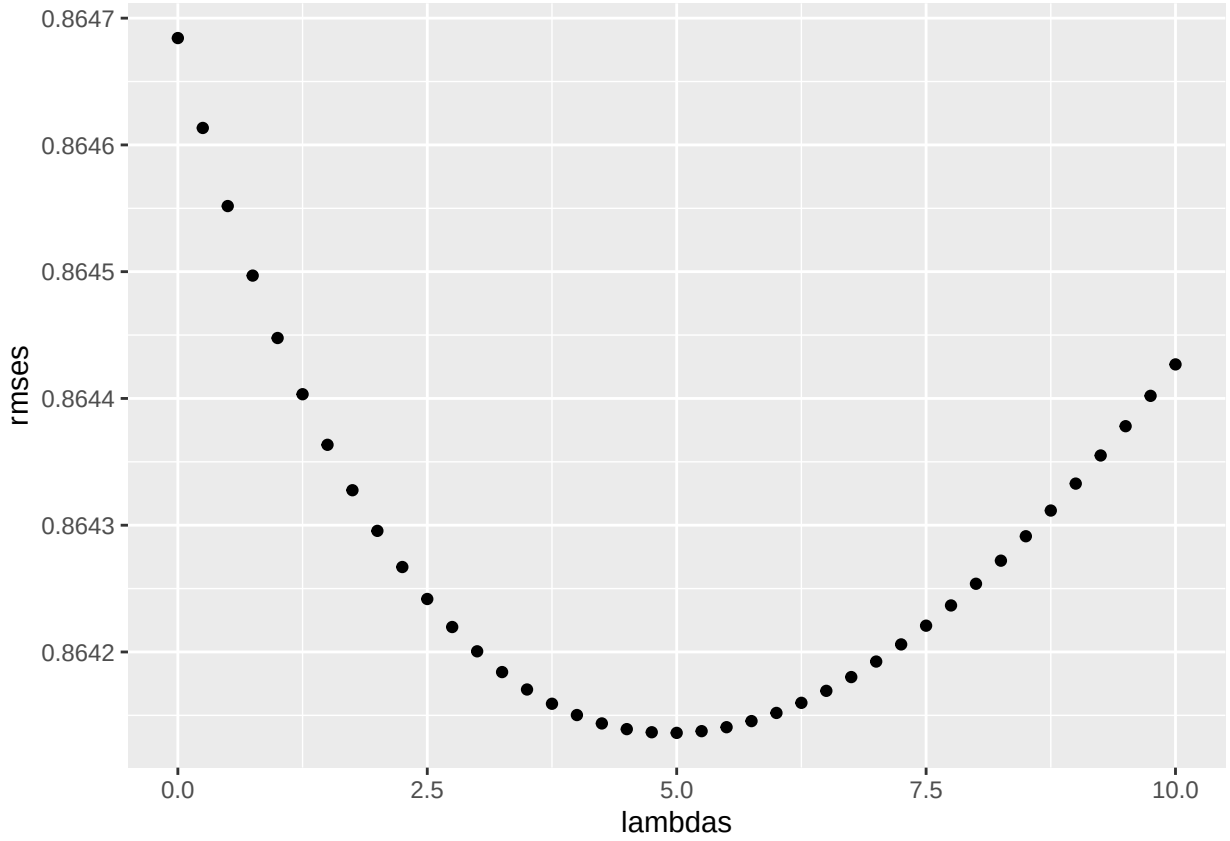
```
  group_by(movieId) %>%
  summarise(b_hat_i = sum(rating - mu_hat)/(n()+lambda), n_i = n())

predicted_ratings <- test_data %>%
  left_join(movie_regularised_averages, by = "movieId") %>%
  mutate(pred = mu_hat + b_hat_i) %>%
  pull(pred)
RMSE(predicted_ratings, test_data$rating)

model_4_rmse <- RMSE(test_data$rating, predicted_ratings)
```

2.5 Regularised Movie and User Effect Model

The regularisation model uses penalised least squares and the tuning parameter lambda is calculated using cross validation. The lambda leading to the lowest RMSE is used on the model as shown on the figure below.



The lambda value with the lowest RMSE is **5**

```
lambda <- 5  #This value is derived from the cross validation.

mu_hat <- mean(train_data$rating)
regularised_user_averages <- train_data %>%
  left_join(movie_averages , by="movieId") %>%
  group_by(userId) %>%
  summarise(b_hat_u = sum(rating - mu_hat - b_hat_i)/(n()+lambda), n_i = n())
```

```
predicted_ratings <- test_data %>%
  left_join(movie_regularised_averages, by = "movieId") %>%
  left_join(regularised_user_averages, by ="userId") %>%
  mutate(pred = mu_hat + b_hat_i + b_hat_u) %>%
  pull(pred)
RMSE(predicted_ratings, test_data$rating)

model_5_rmse <- RMSE(test_data$rating, predicted_ratings)
```

2.6 Matrix Factorisation Model

The matrix factorisation model was created using the Recosystem Package. For the matrix factorisation model, the original dataset format had to be converted to an object of class DataSource to enable the use of the Recosystem package. The methodology used in the matrix factorisation is based on Recosystem documentation page *(https://rdocumentation.org/packages/recosystem/versions/0.5)*

2.6.1 Create a model object using train and test datasets **Train set:** This is an object of class DataSource originaing from the **train_data** dataset and the Recosystem function used is **data_memory()** *(https://www.rdocumentation.org/packages/recosystem/versions/0.5)*.

```
train_dataset <- with(train_data,data_memory(user_index = userId,
                                             item_index = movieId,
                                             rating = rating,
                                             index1 = TRUE))
```

Test set: This is an object of class DataSource originating from **test_data** dataset and the Recosystem function used for this is **data_memory()** *(https://www.rdocumentation.org/packages/recosystem/versions/0.5)*.

```
test_dataset <- with(test_data, data_memory(user_index = userId,
                                            item_index = movieId,
                                            rating = rating,
                                            index1 = TRUE))
```

The model object **r** was created with a Reference Class object in R using the **Reco()** function *(source https://www.rdocumentation.org/packages/recosystem/versions/0.5)*

```
r = Reco()
```

2.6.2 Tuning parameters This project used the Recosystem default tuning parameters to simplify the analysis. The default values have been selected as per Rdocumentation webpage *(https://www.rdocumentation.org/packages/recosystem/versions/0.5/topics/tune)*. Therefore, the **$tune()** function is not necessary on this occasion and was not used for the analysis.

2.6.3 Train the model The model is trained by using the Recosystem **$train()** function.

```
r$train(train_dataset)
```

2.6.4 Compute predicted values This is the same analysis used previously to calculate the RMSE and to compile it to the result table shown in the next section.

```
predicted_ratings = r$predict(test_dataset, out_memory())
model_6_rmse <- RMSE(test_data$rating, predicted_ratings)
```

3. Results

The following table shows the RMSE achieved by the models trained and tested.

Method	RMSE
1. Average Calculation Model	1.0600537
2. Movie effect Model	0.9429615
3. Movie + User effect Model	0.8646843
4. Regularised Movie effect Model	0.9429370
5. Regularised Movie + User effect Model	0.8641788
6. Matrix Factorisation (Recosystem) model	0.8326744

By looking at the different models used so far, it can be deduced that the best performing model is the one using **matrix factorisation** with the Recosystem package. This model will be used to test the result using the **validation set** (final hold-out test set) as follows.

3.1 Create a model object using edx and validation datasets

Train set: This is an object of class DataSource originating from the **edx** dataset and the Recosystem function used for this is **data_memory()**.

```
edx_dataset <- with(edx, data_memory(user_index = userId,
                                     item_index = movieId,
                                     rating = rating,
                                     index1 = TRUE))
```

Validation set: This is an object of class DataSource originating from validation dataset and the Recosystem function used for this is **data_memory()**

```
validation_dataset <- with(validation, data_memory(user_index = userId,
                                                   item_index = movieId,
                                                   rating = rating,
                                                   index1 = TRUE))
```

The model object **r** was created with a Reference Class object in R using the **Reco()** function

```
r = Reco()
```

3.2 Tuning parameters

As stated previously, this project used the Recosystem default tuning parameters to simplify the analysis. Therefore, the **$tune()** function was not used for the analysis.

3.3 Train the model

The model is trained by using the Recosystem **$train()** function.

```
r$train(edx_dataset)
```

3.4 Compute predicted values

```
predicted_ratings = r$predict(validation_dataset, out_memory())
model_7_rmse <- RMSE(validation$rating, predicted_ratings)
```

The following table shows a comparison of the models used as well as the final RMSE achieved by the best performing model which was the one using **matrix factorisation**.

Method	RMSE
1. Average Calculation Model	1.0600537
2. Movie effect Model	0.9429615
3. Movie + User effect Model	0.8646843
4. Regularised Movie effect Model	0.9429370
5. Regularised Movie + User effect Model	0.8641788
6. Matrix Factorisation (Recosystem) model	0.8326744
7. Final Validation Matrix Factorisation (Recosystem) model	0.8324166

As can be seen in the table above, the final **RMSE** achieved by the best performing model (Matrix Factorisation using Recosystem) was **0.8324166** which was evaluated using the **valuation** dataset

4. Conclusion

4.1 Summary

The project consisted of 3 main parts:

4.1.1 Data Exploration and Dataset Creation This step included the review of the Movielens data provided as part of the course handout which was sourced from Grouplens. The Datasets necessary for the data analysis and Machine Learning model creation were obtained from the Movielens data. The datasets created included a partition of the Movielens dataset to create the validation dataset. Additionally the training and test sets were created from the partitioned Movielens data.

4.1.2 Data Analysis Following the creation of the validation, training and test datasets, 6 different models were trained and tested using different techniques including regularisation and matrix factorisation. The models calculated the Root Mean Square Error (RMSE) which was used as evaluation metrics in order to compare the models and to select the best performing model.

4.1.3 Best performing model selection The best performing model based on the test set was the one using matrix factorisation which was modeled using the Recosystem package. This model was then evaluated using the final valuation dataset which produced a **RMSE** of **0.8324166**, leading to a satisfactory result.

4.2 Limitations

The models evaluated were simple models using in most cases default parameters, this was more relevant for the matrix factorisation model. The main driver for this was the preference of the author of this report to provide simple and less complex models. The main objective has been to reach the goal of selecting the best performing model among the different Machine Learning modeling techniques. Should larger datasets become necessary, the model could take a considerable amount of time to run, so it is preferable to limit the size of the datasets to a level similar to the one used for this project.

4.3 Future work

It would be beneficial to improve the models evaluated so more tuning parameters are used specially on the matrix factorisation model. This could lead to more robust results and to enable the use of even larger and more complex datasets.

Although there is room for improvement regarding the models, the author of this report believed that the main objective of the project has been achieved and the final result exceeded the expectation.