Report Capstone Project MovieLens

Isabel Kornmann

.

1. Introduction

The goal of this project is to develop a Machine Learning algorithm that can, after being tuned and trained, predict a movie rating for a specific user and movie on an unknown dataset with a certain precision.

In practice these type of algorithms (Recommendation systems) often use ratings that e.g. a user of a movie streaming platform or a customer of an onlineshop has given a certain movie or product, to predict potential other movies or products that the user/customer might like.[1] Precise recommendation system algorithms therefore provide the potential for big financial gains for the companies that can use their data effectively for accurate predictions and recommendations.

The analysis of this project will be based on the a subset of the MovieLens dataset. The MovieLens database is generated by the GroupLens research lab[2] and contains 27 million ratings for 58,000 movies by 280,000 users.[3] This project will only use a subset of the MovieLens dataset called MovieLens 10M dataset, which contains 10 million ratings from 10,000 movies by 72,000 users.[4]

The report is structured as follows: The first step in the analysis is the data preparation which includes the data import and the creation of 3 subsets (Train, validation and test). This is followed by an exploration of the data and some data cleaning. Then the evaluation criteria for performance as well as the algorithms used in this project will be introduced briefly. Subsequently the training set will be used to train different models which will thereafter be applied to the validation set to see which performs the best. The best performing algorithm will then be trained and used to predict movie ratings for the (unknown) test set. A summary of the results and the performance of the different algorithms as well as a discussion about limitations and ideas for further investigation conclude this report.

2. Methods and Analysis Section

This section describes how the datasets for the ensuing analysis are created by splitting the downloaded data into a training, validation and test set. This data preparation step is needed to enable the training and evaluation of different algorithms. This subsequent section contains an exploration of the data. Furthermore, the different algorithms used in this project are introduced and the motivation for the choice of these models is outlined. Moreover, the evaluation criteria for algorithm performance is established.

[1] http://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#recommendation-systems
[2] https://grouplens.org/
[3] https://grouplens.org/datase/movielens/latest/
[4] https://grouplens.org/datasets/movielens/10m/

2.1 Data Preparation

2.1.1. Download and Preparation of MovieLens Dataset

First the data, that will be used in this analysis, will be downloaded and the packages required are installed and loaded. Furthermore, the downloaded dataset will be prepared for the use in this machine learning project. The processed data is saved in the `movielens` dataframe.

```
# Install necessary packages
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(recosystem)) install.packages("recosystem", repos =
                                         "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos =
                                           "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos =
                                            "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos =
                                              "http://cran.us.r-project.org")


# Load necessary packages
library(tidyverse)
library(caret)
library(recosystem)
library(knitr)
library(ggplot2)
library(data.table)


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

# Timeout time for download increased to 120s instead of default of
# 60s to account for longer time needed to download files
options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip",
                dl)
```

```
ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"),
                                   simplify = TRUE), stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"),
                                  simplify = TRUE), stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

movielens <- left_join(ratings, movies, by = "movieId")
```

2.1.2 Creating edx, training, validation, and test set

In the next step the `movielens` dataset will be split into a training set for the final model (`edx`) and the test set on which the final algorithm will be evaluated (`final_holdout_test`). The two datasets will consist of 90% and 10% of the `movielens` data respectively.

```
# Final_holdout_test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1,
                                  p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final_holdout_test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

```
# Add rows removed from final_holdout_test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

# Remove objects not needed anymore
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

To check Whether the datasets were created correctly we divide the dimensions of the respective datasets and see that the targeted 90/10 split is achieved.

```
# Check if datasets were created correctly (should be approx. 10%)
dim(final_holdout_test) / dim(edx)
```

```
## [1] 0.1111103 1.0000000
```

To get a true impression on how the final algorithm developed in this project performs, the `final_holdout_test` set has to remain unknown and can neither be used to train nor to test the algorithms developed during the course of this project. Therefore an additional dataset needs to be created to evaluate the performance of various models analyzed in this project. To achieve this the `edx` dataset will be split into a training set `edx_train` and a validation set `val_set`. The split will again be 90/10 respectively.

```
# Creating validation set that will be 10% of remaining edx data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
val_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1,
                                 list = FALSE)
edx_train <- edx[-val_index,]
temp <- edx[val_index,]

# Make sure userId and movieId are in edx/final_holdout_test set and also
# in validation set
val_set <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from val_set set back into edx set
removed <- anti_join(temp, val_set)
edx <- rbind(edx, removed)

# Remove objects
rm(val_index, temp, removed)
```

To check if the datasets were created correctly we divide the dimensions of the respective datasets and see that the targeted 90/10 split is achieved.

```
# Check if datasets are created correctly (should be approx. 10%)
dim(val_set) / dim(edx_train)
```

```
## [1] 0.1111113 1.0000000
```

After the final model is determined by the best performance on the validation set `val_set`, the model is trained on the entire `edx` data set and the performance will be evaluated on the `final_holdout_test` set.

2.2 Data Exploration

To get an understanding of the dataset used in this project, this section will focus on exploration of the `edx` dataset. First we will examine the structure and the variables of the dataframe.

```
# Look at structure of edx
str(edx, width=60, strict.width = "cut")
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 355 356 362 364 370..
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392..
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" ""..
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thrille"..
```

```
# Get names of variables in edx
variable.names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

The dataframe `edx` consists of approximately 9 million observations for the 6 variables:

- userId (int)
- movieId (int)
- rating (num)
- timestamp (int)
- title (char)
- genres (char)

Further analysis reveals that the number of unique users in the dataset is 69878 and the number of distinct movies is 10677. Since the number of distinct users is larger than the number of movies, this suggests that not every user has rated every movie.

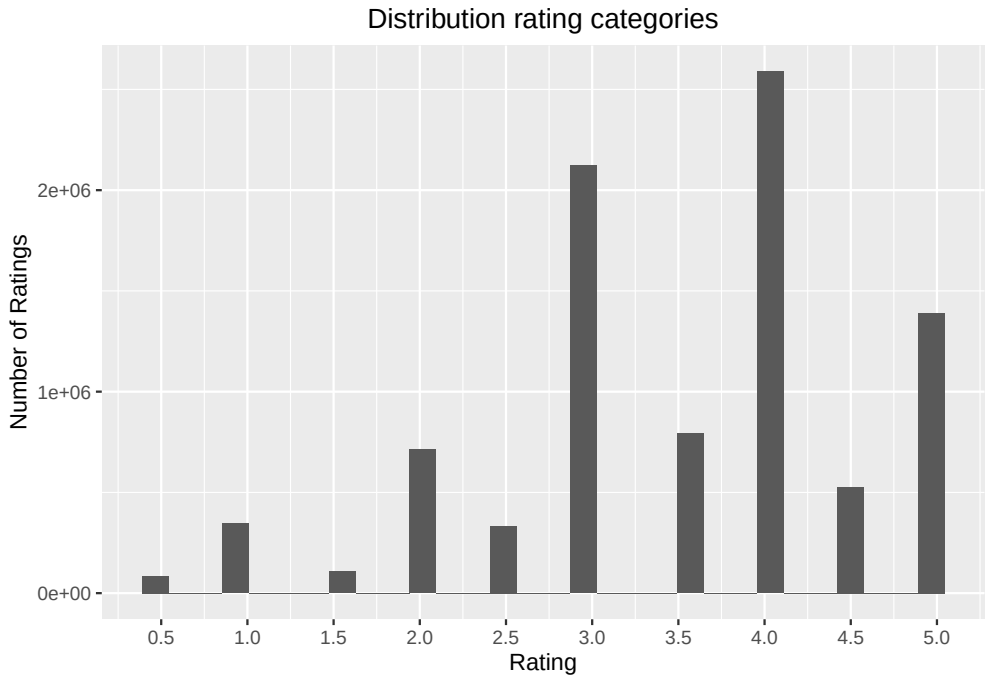
```
# Number of unique users that provided ratings and how many unique movies
# were rated
edx %>% summarize(n_users = n_distinct(userId),
                  n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

The analysis in this project and the algorithm development will focus on the variables rating, movieId and userId and therefore these 3 variables will also be at the center of the subsequent data exploration.

Plotting the number of ratings that fall into one of the rating categories shows that the ratings for movies range from 0.5 to 5 in 0.5 increments. The majority of ratings is either 3 or 4. Furthermore, there are more ratings that end on 0 than on .5 . Overall the histogram shows a tendency towards better ratings (3 and upwards).

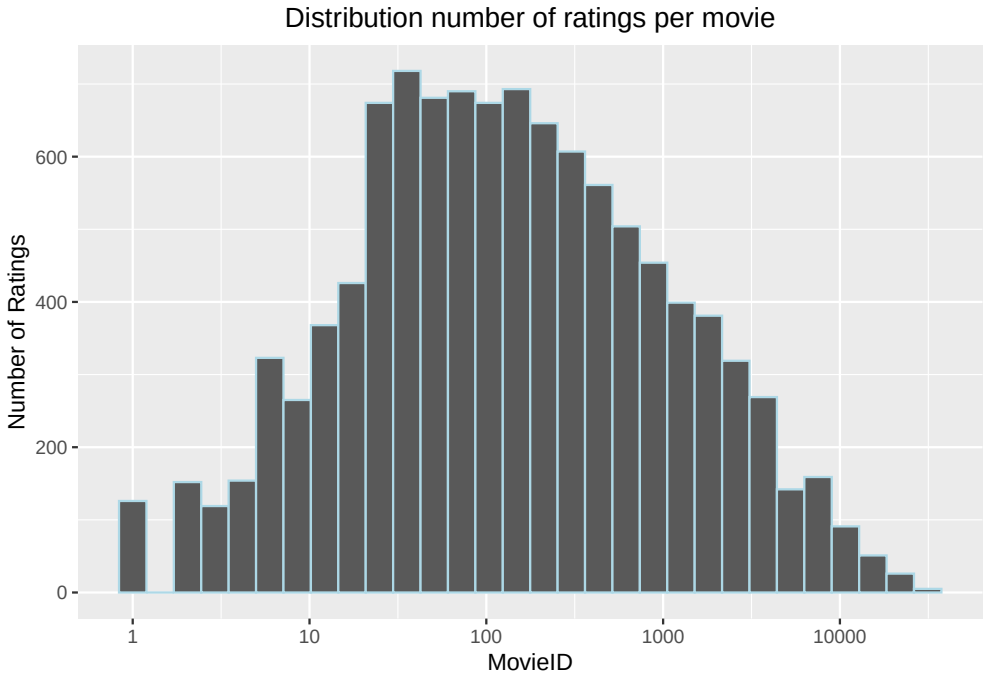
```
# Histogram of number of ratings per rating category
edx %>% group_by(rating) %>%
  ggplot(aes(x=rating)) +
  geom_histogram() +
  xlab("Rating") +
  ylab("Number of Ratings") +
  scale_x_continuous(breaks=seq(0,5,0.5)) +
  ggtitle("Distribution rating categories") +
  theme(plot.title = element_text(hjust=0.5))
```



The shape of the histogram of the number of ratings per movie demonstrates that there are some movies that are rated more often than others.

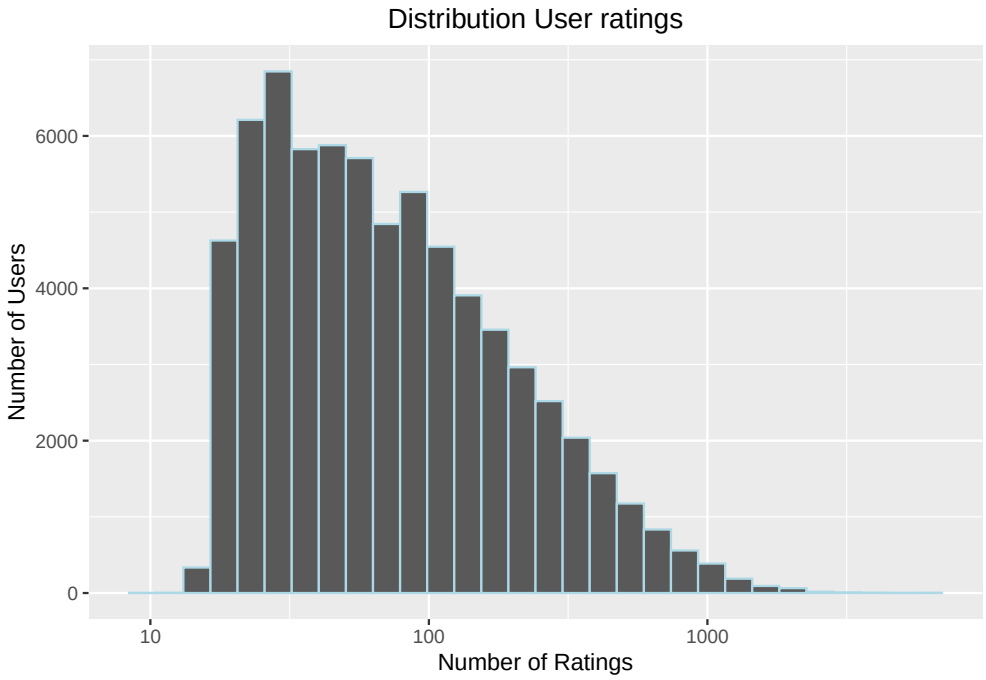
```
# Histogram of number of ratings per movie
edx %>%
```

```
  count(movieId)%>%
  ggplot(aes(n)) +
  geom_histogram(color = "lightblue") +
  scale_x_log10() +
  ggtitle("Distribution number of ratings per movie") +
  xlab("MovieID") +
  ylab("Number of Ratings")+
  theme(plot.title = element_text(hjust=0.5))
```



The distribution of the number of times users have rated any movie is right skewed and indicates that the majority of users have rated between 30-100 movies. There are visibly no users that have rated less than 10 movies and the graph shows a significant amount of users in the edx data that have rated between 100 and 1000 movies. The graph also depicts that users differ in how many movies they rate.

```
# Histogram distribution number of user ratings
edx %>% count(userId)%>%
  ggplot(aes(n)) +
  geom_histogram(color="lightblue")+
  scale_x_log10()+
  xlab("Number of Ratings") +
  ylab("Number of Users") +
  ggtitle("Distribution User ratings") +
  theme(plot.title = element_text(hjust=0.5))
```



2.3 Data Cleaning & Matrix Conversion

The analysis of this project is limited to the use of the variables movieId and userId for the prediction of ratings. Therefore we will only select these 3 variables (movieId, userId, rating) from the datasets (`edx`, `edx_train`) and convert the dataset with these variable into a matrix to facilitate the subsequent training and development of prediction models. The endresult is a matrix with the userIds as rows and the movieIds as columns with ratings being the data in the cells.

For the `edx` dataset:

```
# Only use moveId, userId, rating as predictors
# Create pivot_wider matrix with users in rows and movies in columns +
# ratings in cells
y_edx <- select(edx, movieId, userId, rating) %>%
  pivot_wider(names_from = movieId, values_from = rating)
rnames <- y_edx$userId
y_edx <- as.matrix(y_edx[,-1])
rownames(y_edx) <- rnames

# Map movie ids to titles
movie_map <- edx %>% select(movieId, title) %>%
  distinct(movieId, .keep_all = TRUE)
```

For the `edx_train` dataset:

```r
# Only use movieId, userId, rating as predictors
# Create pivot_wider matrix matrix with users in rows and movies in columns +
# ratings in cells
y_edx_train <- select(edx_train, movieId, userId, rating) %>%
  pivot_wider(names_from = movieId, values_from = rating)
rnames <- y_edx_train$userId
y_edx_train <- as.matrix(y_edx_train[,-1])
rownames(y_edx_train) <- rnames

# Map movie ids to titles
movie_map <- edx_train %>% select(movieId, title) %>%
  distinct(movieId, .keep_all = TRUE)
```

2.4 Evaluation Criteria

To evaluate the performance of an algorithm, first on the validation and later for the final model on the test set, we use the residual mean squared error (RMSE). The RMSE is defined as

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

with $y_{u,i}$ as the rating from movie i by user u, $\hat{y}_{u,i}$ the respective prediction and N denoting the number of user/movie combinations.[5] Since for the prediction of movie ratings the RMSE denotes the typical error we make when predicting a movie rating, the algorithm with the smallest RMSE on the validation set `val_set` is considered to be the best performing algorithm and will subsequently be trained on the `edx` dataset and used to predict ratings for the `final_holdout_test` set.

2.5 Modeling / Models used in the analysis

In this section we will briefly introduce the models that will be trained on the `edx_train` dataset and whose performance will be evaluated on the validation set `val_set`.

2.5.1 Baseline Model/Average

The RMSE of the first model will serve as the baseline model performance, which we will try to improve by using more sophisticated algorithms for the prediction of the ratings. With the first model we predict the same rating for all movies and users and that all variation in ratings is explained by random error :

[5] http://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#netflix-loss-function

$$Y_{u,i} = \mu + \epsilon_{i,u}$$

Where $\epsilon_{i,u}$ is the error term, $Y_{u,i}$ the predicted rating for user u for movie i and μ is the mean. To approximate the "true" parameter μ we take the average of all ratings.

2.5.2 Movie Effects

The first change to the baseline model is to include a Movie effect b_i which leads to our new model:[6]

$$Y_{u,i} = \mu + b_i + \epsilon_{i,u}$$

The motivation for including a Movie specific effect is simply that from experience there are movies that are better received by an audience than others and this popularity of certain movies should be included in our model. Since we transformed the dataset on which we want to train our algorithm (`edx_train`) into a matrix, with each column containing all ratings for a specific movie, we can calculate the movie effect directly by taking the mean over each column.

2.5.3 Movie & User Effects

In addition to the Movie Effect we will also include a User Effect b_u to account for a person's inherent predisposition to either rate all movies better or worse than the average viewer. The inclusion of a User Effects leads to this model:[7]

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

In our analysis we approximate the User effect as the average of the difference between the observation $y_{u,i}$ and the estimated rating average $\hat{\mu}$ and the estimated movie effect $\hat{b}_i$:

$$\hat{b}_u = \frac{1}{N}\sum_{i=1}^{N}(y_{u,i} - \hat{\mu} - \hat{b}_i)$$

By applying this formula to the matrix `y_edx_train` we can estimate the User effect and increase the prediction power of our model.

2.5.4 Regularization and Penalized Least Squares

Furthermore, we can improve our prediction model by penalizing large estimates that are formed using small sample sizes. This process is called Regularization. In our model we use a penalized least squares regression algorithm. To get the optimal parameters we minimize

[6] http://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#modeling-movie-effects
[7] http://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#user-effects

an equation that includes a penalty term. Through calculus we derive that the movie effect estimate that minimizes the RMSE equation is given by[8]

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

with n_i being the number of ratings made for movie i and tuning parameter λ. With the optimal λ we estimated $\hat{b}_i$ and calculate the parameter $\hat{b}_u$ to also account for the user effect.

2.5.5 Matrix Factorization with the 'Recosystem' Package

An additional algorithm that will be used in an attempt to minimize the RMSE of our model is Matrix Factorization with the 'Recosystem' Package.[9]

The `recosystem` package can be used to train a recommender system using matrix factorization. The recommender system will be used to predict the unknown entries in the rating matrix `y_edx_train` based on the observed Movieratings by users using matrix factorization.

One reason this package was included in the analysis is that the 'recosystem' package can take advantage of multicore CPUs to speed up the computation time needed to train the algorithm significantly.

To create a Recommender Model with 'recosystem' the following steps need to be taken:[10],[11]

1. Specify the vector for the user and item indices of the rating scores and the vector of observed entries in the rating matrix.
2. Create a model object by calling `Reco()`.
3. Choose your method to select the best tuning parameters and a set of possible values with the `$tune()` option.
4. Train your model on your training set with the tuning parameters specified by `tune()` by calling `$train`.
5. Use `$predict` to compute the predicted values.

[8] http://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#penalized-least-squares
[9] https://cran.r-project.org/package ==recosystem
[10] https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html
[11] https://cran.r-project.org/web/packages/recosystem/recosystem.pdf

3. Results

The algorithms will be trained on the `edx_train` dataset and then the RMSE of the algorithms will be calculated in the validation set `val_set`. Based on the results, the algorithm with the lowest RMSE on the `val_set` will be chosen to be trained on the full `edx` dataset and then used to predict the Movieratings on the `final_holdout_test` set.

3.1 Baseline Model/Average

```
### Baseline Model
# Simple model -> predict same rating for all movies regardless of user
mu <- mean(y_edx_train, na.rm = TRUE)

# Evaluation of model on val_set
baseline_rmse_val <- RMSE(val_set$rating, mu)
baseline_rmse_val
```

```
## [1] 1.060056
```

```
# Collect the results of the model performance on val_set in Table
results_table <- tibble(Model = "Baseline/Average", RMSE = baseline_rmse_val)
results_table
```

```
## # A tibble: 1 x 2
##   Model             RMSE
##   <chr>            <dbl>
## 1 Baseline/Average  1.06
```

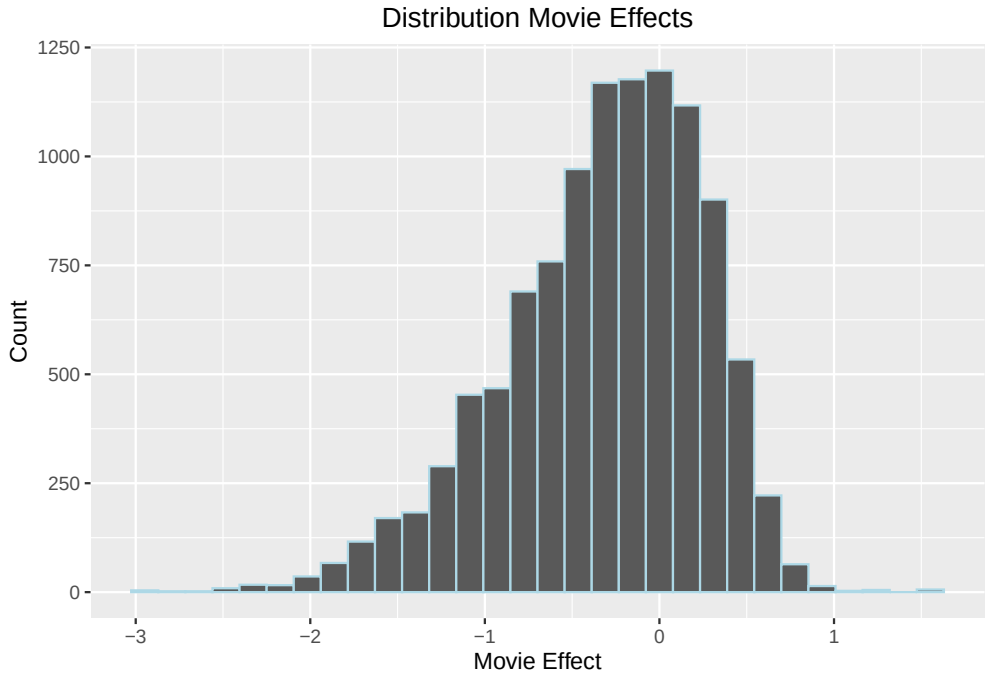
The first RMSE that we achieved with our Baseline/Average model is 1.060056 on the valdation set `val_set`. This will be the benchmark that the subsequent algorithms have to undercut to be classified as the superior model.

3.2 Movie Effects

```
# Model with Movie Effects
b_i <- colMeans(y_edx_train - mu, na.rm = TRUE)
fit_movies <- data.frame(movieId = as.integer(colnames(y_edx_train)),
                         mu = mu, b_i = b_i)
```

Plotting the distribution of estimates for the Movie Effects show that the these estimates vary drastically and therefore an inclusion in the predicting algorithm should enhance the model performance.

```
# Histogram of Movie effects
as.data.frame(b_i) %>% ggplot(aes(b_i))+
  geom_histogram(color="lightblue") +
  xlab("Movie Effect")+
  ylab("Count")+
  ggtitle("Distribution Movie Effects")+
  theme(plot.title = element_text(hjust=0.5))
```




```
# Evaluation of model on val_set
RMSE_Movie <- left_join(val_set, fit_movies, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  summarize(rmse = RMSE(rating, pred, na.rm=TRUE))
RMSE_Movie
```

```
##       rmse
## 1 0.9429615
```

```
results_table <- bind_rows(results_table,
                           tibble(Model= "Movie Effects",
                                RMSE = as.numeric(RMSE_Movie)))
results_table
```

```
## # A tibble: 2 x 2
##   Model             RMSE
##   <chr>            <dbl>
## 1 Baseline/Average 1.06
## 2 Movie Effects    0.943
```

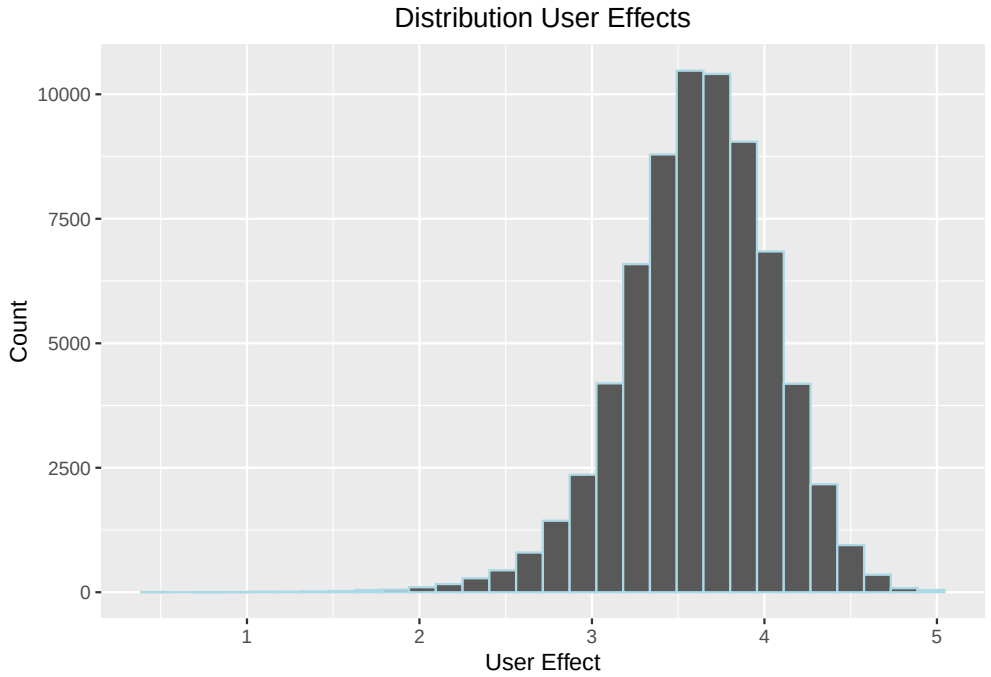
As expected reduces the inclusion of an estimate for the movie effect our RMSE substantially by around 0.12 to an RMSE for the Movie Effects model of 0.9429615 on the validation set `val_set`.

3.3 Movie & User Effects

```
# With User Effects & Movie Effects
b_u <- rowMeans(y_edx_train, na.rm = TRUE)
```

Plotting the estimates of the User effects shows that there is substantial variability between the users when rating a movie. This suggests an inclusion of an User Effect term in our model could further reduce our RMSE.

```
# Histogram of User Effects
as.data.frame(b_u) %>% ggplot(aes(b_u))+
  geom_histogram(color="lightblue") +
  xlab("User Effect")+
  ylab("Count")+
  ggtitle("Distribution User Effects")+
  theme(plot.title = element_text(hjust=0.5))
```




```
b_u <- rowMeans(sweep(y_edx_train - mu, 2, b_i), na.rm = TRUE)

fit_users <- data.frame(userId = as.integer(rownames(y_edx_train)), b_u = b_u)
```

```
# Evaluation of model on val_set
RMSE_MovieUser <- left_join(val_set, fit_movies, by = "movieId") %>%
  left_join(fit_users, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  summarize(rmse = RMSE(rating, pred, na.rm=TRUE))
RMSE_MovieUser
```

```
##        rmse
## 1 0.8646844
```

```
results_table <- bind_rows(results_table,
                           tibble(Model= "Movie and User Effects",
                                  RMSE = as.numeric(RMSE_MovieUser)))
results_table
```

```
## # A tibble: 3 x 2
##   Model                  RMSE
##   <chr>                 <dbl>
## 1 Baseline/Average      1.06
## 2 Movie Effects         0.943
## 3 Movie and User Effects 0.865
```

As already hinted through the graphical examination of the User Effects, an additional term to account for User Effects in our model further reduced our RMSE by 0.078 to 0.8646844.

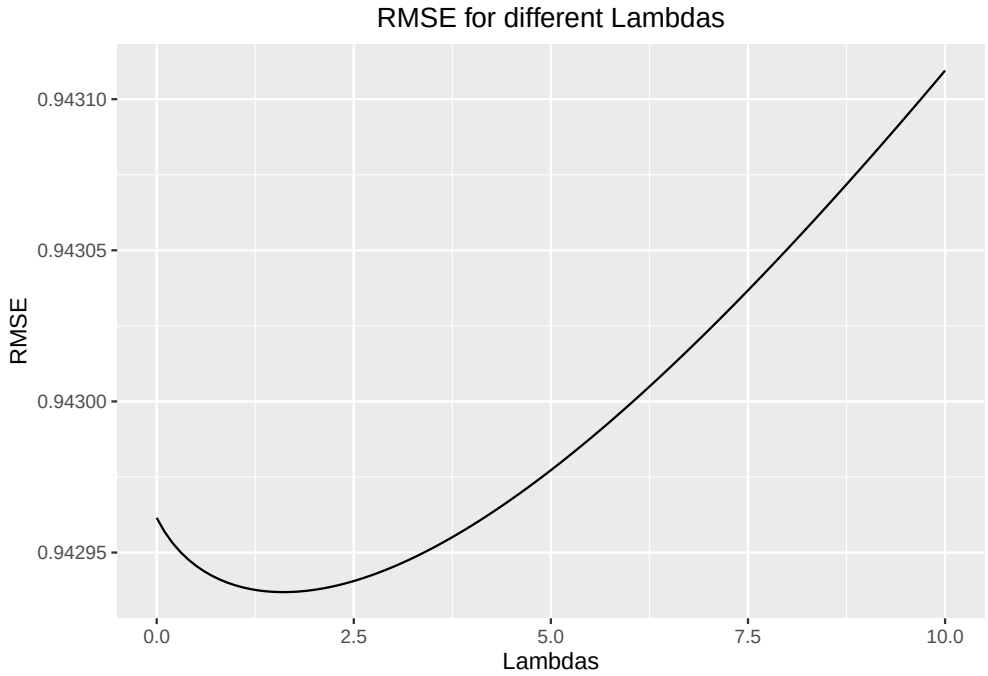
3.4 Regularization and Penalized Least Squares

```
# Regularization parameter
lambdas <- seq(0, 10, 0.1)

# Regularization model with Penalized Least Squares
sums <- colSums(y_edx_train - mu, na.rm = TRUE)
n <-  colSums(!is.na(y_edx_train))
fit_movies$n <- n

# Train & Tune model
rmses <- sapply(lambdas, function(lambda){
  b_i <-  sums / (n + lambda)
  fit_movies$b_i <- b_i
  left_join(val_set, fit_movies, by = "movieId") %>% mutate(pred = mu + b_i) %>%
    summarize(rmse = RMSE(rating, pred, na.rm=TRUE)) %>%
    pull(rmse)
```

```
})
```

Plotting the parameter λ used in Regularization against the performance evaluation criteria RMSE shows graphically that there exists a value for lambda that minimizes the RMSE of the model. The goal of training and tuning of the model therefore should be to find the value for λ that minimizes the RMSE of the model.

```
# Plot RMSE dependent on lambdas
tibble(Lambdas=lambdas, RMSE=rmses) %>%
  ggplot(aes(Lambdas, RMSE)) +
  geom_line() +
  ggtitle("RMSE for different Lambdas")+
  theme(plot.title = element_text(hjust=0.5))
```



Using the λ that minimizes the RMSE to calculate the Movie and User effects for our model:

```
# Chose Lambda that minimizes RMSE
lambda <- lambdas[which.min(rmses)]

# Calculate Movie effect
fit_movies$b_i_reg <- colSums(y_edx_train - mu, na.rm = TRUE) / (n + lambda)

# Calculate User Effect
```

```
fit_users$b_u <- rowMeans(sweep(y_edx_train - mu, 2, b_i), na.rm = TRUE)

# Calculate RMSE with Tuned parameters
RMSE_reg <- left_join(val_set, fit_movies, by = "movieId") %>%
  left_join(fit_users, by = "userId") %>%
  mutate(pred = mu + b_i_reg + b_u) %>%
  summarize(rmse = RMSE(rating, pred, na.rm = TRUE))
RMSE_reg
```

```
##        rmse
## 1 0.8646081
```

```
results_table <- bind_rows(results_table,
                           tibble(Model= "Regularization",
                                  RMSE = as.numeric(RMSE_reg)))
results_table
```

```
## # A tibble: 4 x 2
##   Model                    RMSE
##   <chr>                   <dbl>
## 1 Baseline/Average        1.06
## 2 Movie Effects           0.943
## 3 Movie and User Effects 0.865
## 4 Regularization          0.865
```

The model that uses Regularization further decreased the RMSE achieved with previous models but only slightly to a RMSE of 0.8646081.

3.5 Matrix Factorization with 'Recosystem' Package

Although the 'recosystem' package is chosen because it has a relatively short computation time when training the algorithm, the code may still take a few hours to run.

First specify the data needed for 'recosystem' (user_index, item_index, rating). Then create the model object `r`. Afterwards select the tuning parameters that should be used in the training of the algorithm. In this case we use the default parameter except for `nthread` and `niter`. `nthread` defined the number of threads used for parallel computing. This is being changed from the default 1 to 4 to decrease the computation time of the code. `niter` specifies the number of iterations and is decreased from the default 20 to 10 to also decrease computation time. Then the algorithm is trained and subsequently used to predict movie ratings. Afterwards the RMSE for the algorithm on the validation set `val_set` is calculated and returned.

```
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier

# Specify data for Recosystem
```

```
edx_data <- with(edx_train, data_memory(user_index = userId,
                                        item_index = movieId,
                                        rating = rating))
val_data <- with(val_set, data_memory(user_index = userId,
                                      item_index = movieId,
                                      rating=rating))

# Create the model
r <- Reco()

# Select the option for the model
opts <- r$tune(edx_data, opts = list(nthread = 4, niter= 10))

# Training of the algorithm
r$train(edx_data, opts= c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9755   1.0916e+07
##    1       0.8764   8.9994e+06
##    2       0.8454   8.3780e+06
##    3       0.8253   8.0109e+06
##    4       0.8096   7.7446e+06
##    5       0.7984   7.5653e+06
##    6       0.7897   7.4287e+06
##    7       0.7825   7.3228e+06
##    8       0.7763   7.2333e+06
##    9       0.7709   7.1607e+06
##   10       0.7662   7.0990e+06
##   11       0.7621   7.0464e+06
##   12       0.7585   6.9998e+06
##   13       0.7554   6.9613e+06
##   14       0.7526   6.9257e+06
##   15       0.7502   6.8965e+06
##   16       0.7479   6.8682e+06
##   17       0.7459   6.8450e+06
##   18       0.7440   6.8246e+06
##   19       0.7423   6.8036e+06
```

```
# Create Predicted values
Pred_reco <- r$predict(val_data, out_memory())

#RMSE
rmse_reco <- RMSE(val_set$rating, Pred_reco)
rmse_reco
```

```
## [1] 0.7906471
```

```
results_table <- bind_rows(results_table,
                           tibble(Model= "Matrix Factorisation (Recosystem)",
                                  RMSE = as.numeric(rmse_reco)))
results_table
```

```
## # A tibble: 5 x 2
##   Model                              RMSE
##   <chr>                             <dbl>
## 1 Baseline/Average                  1.06
## 2 Movie Effects                     0.943
## 3 Movie and User Effects            0.865
## 4 Regularization                    0.865
## 5 Matrix Factorisation (Recosystem) 0.791
```

The model that uses the 'Recosystem' package with Matrix factorization achieves with 0.791476 a significantly lower RMSE compared to the models we looked at before.
3.6 Train and test the performance of the final model

Since the model Matrix Factorization with the 'Recosystem' package reached the lowest RMSE of all models on the validation set `val_set` we will now train this algorithm on the `edx` dataset and test the performance of the model on the `final_holdout_test` set.

```
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier

# Convert to format needed for Recosystem
edx_full_data <- with(edx, data_memory(user_index = userId,
                                       item_index = movieId,
                                       rating = rating))
final_holdout_test_data <- with(final_holdout_test, data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating=rating))

# Create the model object
r_final <- Reco()

# Select the option for the model
opts <- r_final$tune(edx_full_data, opts = list(nthread = 4, niter= 10))

# Training of the algorithm
r_final$train(edx_full_data, opts= c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
```

```
##    0       0.9632   1.1820e+07
##    1       0.8730   9.8809e+06
##    2       0.8411   9.1864e+06
##    3       0.8217   8.7964e+06
##    4       0.8078   8.5452e+06
##    5       0.7968   8.3492e+06
##    6       0.7883   8.2026e+06
##    7       0.7813   8.0934e+06
##    8       0.7756   8.0014e+06
##    9       0.7705   7.9267e+06
##   10       0.7663   7.8640e+06
##   11       0.7627   7.8122e+06
##   12       0.7594   7.7681e+06
##   13       0.7565   7.7246e+06
##   14       0.7540   7.6948e+06
##   15       0.7518   7.6642e+06
##   16       0.7497   7.6377e+06
##   17       0.7478   7.6138e+06
##   18       0.7462   7.5929e+06
##   19       0.7447   7.5723e+06
```

```
# Create Predicted values
Pred_reco_final <- r_final$predict(final_holdout_test_data, out_memory())

#RMSE
rmse_reco_final <- RMSE(final_holdout_test$rating, Pred_reco_final)
rmse_reco_final
```

```
## [1] 0.7888142
```

```
results_table <- bind_rows(results_table,
                           tibble(Model= "Matrix Factorisation (Recosystem) - Final Mode
                                  RMSE = as.numeric(rmse_reco_final)))
results_table
```

```
## # A tibble: 6 x 2
##   Model                                          RMSE
##   <chr>                                         <dbl>
## 1 Baseline/Average                              1.06
## 2 Movie Effects                                 0.943
## 3 Movie and User Effects                        0.865
## 4 Regularization                                0.865
## 5 Matrix Factorisation (Recosystem)             0.791
## 6 Matrix Factorisation (Recosystem) - Final Model 0.789
```

The RMSE of our final model on the `final_holdout_test` set is 0.7889377.

4. Conclusion

4.1 Summary of Analysis

The precise results of the analysis conducted in this project are summarized in the following table:

```
results_table %>% kable("html", digit=5)
```

Model

RMSE

Baseline/Average

1.06006

Movie Effects

0.94296

Movie and User Effects

0.86468

Regularization

0.86461

Matrix Factorisation (Recosystem)

0.79065

Matrix Factorisation (Recosystem) - Final Model

0.78881

It is visible from the table that with each adaptation of our algorithm we were able to improve our model performance and decrease our RMSE. Furthermore, it can be noted, that the best performing algorithm on our validation set, also had a very good performance when used on the test set.
The RMSE of our final model on the `final_holdout_test` set is 0.7889377.
4.2 Limitation of presented analysis

This analysis was only a first attempt to develop an algorithm that can reliably predict movie or any other ratings. The main restriction in this project was posed by the large size of the dataset and the subsequent long computation time when training a model or algorithm. The implementation of many Machine Learning algorithms that use a lot of iterations in their trainig was therefore not possible.

4.3 Future Work

Since the analysis in this report was only focused on developing a model using the two predictors movieId and userId, there are extensive possibilities for further improvement of the models and analysis presented here by designing new algorithms using the variables not used in this project. For example including the variables genre or time from the `movielens` dataset to form prediction about movie ratings could be an angle for further investigation.