edX MovieLens Capstone Project

Fabian Pilz

2023-05-28

Introduction

Project Overview

Why machine learning?

Machine learning is a key technology to help modern business to gain insights from raw data. It advances the business by processing the ever growing amount of data to learn from it, understand patterns, behaviors. The gained information can be used to keep up to date with consumer need and as basis for future business strategies (Pierson, 2021). One of the main challenges is to develop effective and efficient machine learning algorithms. One example is the Netflix Prize, which was an open competition that began on October 2, 2006. The goal was to develop an algorithm to predict user ratings for films and improve their movie Recommendation System (Irizarry, 2019). The prize of $1,000,000 was awarded on September 21, 2009 (Lohr, 2009).

This project

This project is based on the Neflix challenge mentioned above and has the same aim, to predict movie ratings for users. For this purpose we train a linear model to generate predicted movie ratings and quantify the performance of the algorithm by calculating the root mean square error (RMSE. The dataset we use is the `MovieLens 10M Dataset`(https://grouplens.org/datasets/MovieLens/10m/) which is a stable benchmark dataset based on the Netflix challenge mentioned above. It contains a total of of 10 million movie ratings.

Aim of the project

What is the goal?

The goal is to build an algorithm to predict the movie ratings of unknown cases based on known data as accurately as possible.

How do we plan to achieve it?

We split the `MovieLens` dataset a total of three times. First into the `edx` dataset (90 %) and the `final_holdout_test` dataset (10 %). The latter remains untouched until we perform the final evaluation. The `edx` dataset will be split again into two parts, first the `edx_train_set` (90 %) - used for model training - and the `edx_validation_set` (10 %) - used to validate the model and optimize parameters. We evaluate the performance of our algorithm using the root mean squared error. We test different machine learning approaches, predict ratings for the validation set, compare them with the real ones, calculate the RSME to compare the models to one another and improve the machine learning algorithm. We conclude the algorithm development by selecting the model with the lowest RMSE. Ideally the value for RSME should be below 0.86490.

Structure of this report

This report consists of five parts: the introduction has presented the problem as well as the aim of the project, the methods and analysis section describes the dataset, develops preliminary studies and establishes the machine learning models, the results section presents the modeling results and the conclusion gives a brief summary of the report, its limitations and future work.

Methods and Analysis

Data Transformation

First of all, we download the `MovieLens` data, tidy and transform it. Once we have the dataset in a tidy structure, we will split the set 90-10 into two parts, the first part of the set called `edx` (90 % of the `MovieLens` dataset) and the `final_holdout_test` (10 % of the `MovieLens` dataset) which we will use in a final step to evaluate our machine learning model. The latter stays untouched until the final test of our completed model, in an attempt to simulate new data.

```
### create datasets Create edx and final_holdout_test sets
### Note: this process could take a couple of minutes
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")

# Final hold-out test set will be 10% of MovieLens data
```

```
set.seed(1, sample.kind = "Rounding")  # if using R 3.6 or later
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 hold-out 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 hold-out test set back into
# edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, MovieLens, removed)
```

we will only use the `edx` set to train and validate our machine learning model. Therefore, in the next step, we split the `edx` set again into parts, the `edx_train_set` and the `edx_validation_set`. We use the training set to train and the validation set to validate our models and tune parameters using cross-validation. Calculating the RMSE for the predicted values of the validation set gives us an idea about the model performance. The split ratio will again be 90-10, with the `edx_train_set` being 90 % and the `edx_validaiton_set` being 10 % of the `edx` dataset.

```
### split the edx data into a train and a validation set so
### that the final_holdout_test stays untouched create
### indices, ratio 90% train, 10% validation
set.seed(1, sample.kind = "Rounding")  # if using R 3.6 or later
test_index <- createDataPartition(y = edx$rating, times = 1,
    p = 0.1, list = FALSE)
edx_train_set <- edx[-test_index, ]
temp <- edx[test_index, ]

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

# Add rows removed from edx_validation_set back into
# edx_train_set
```

```
removed <- anti_join(temp, edx_validation_set)
edx_train_set <- rbind(edx_train_set, removed)

rm(removed, temp, test_index)
```

Exploratory Data Analysis

Before we build a model to predict movie ratings, we need to explore the dataset to better understand underlying effects and correlations.

```
str(edx, vec.len = 2)
```

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

The `edx` dataset in in tidy format and contains a total of 9,000,055 ratings. A single observation is defined by:

- the user who rated it,
- an ID for the movie,
- the movie title,
- its genre(s),
- the time(point) of the rating and finally
- the rating itself.

A look at the summary statistics shows that there are no missing values.

```
summary(edx)
```

```
##      userId        movieId          rating       timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.50   Min.   :7.90e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.00   1st Qu.:9.47e+08
##  Median :35738   Median : 1834   Median :4.00   Median :1.04e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.51   Mean   :1.03e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.00   3rd Qu.:1.13e+09
##  Max.   :71567   Max.   :65133   Max.   :5.00   Max.   :1.23e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

We can count individual entries in each column. There are 10,677 different movies in the dataset rated by 69,878 users. Ten different ratings are possible and since a movie can be assigned to more than one genre, there are 797 different genre combinations.

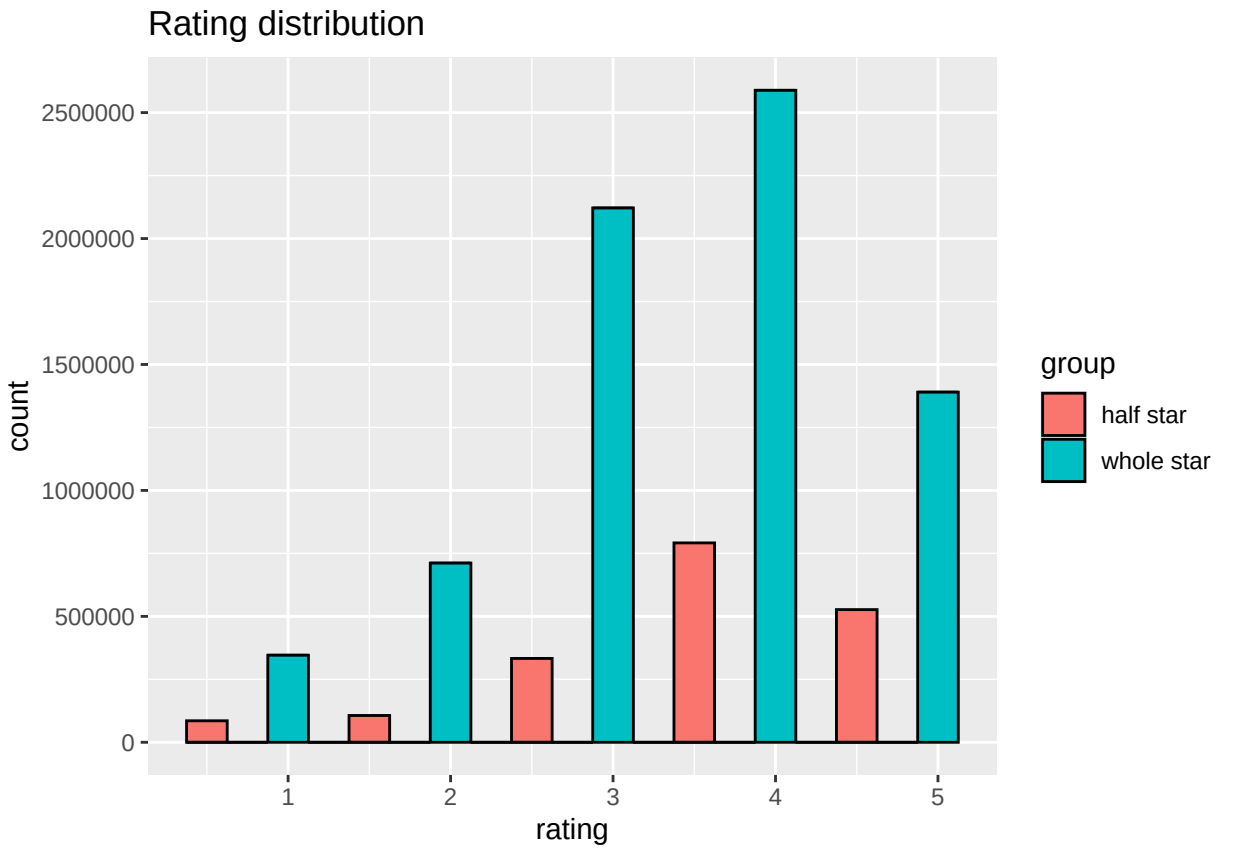
```
edx %>%
    select(-title, -timestamp) %>%
    map_df(. %>%
        n_distinct())
```

```
## # A tibble: 1 x 4
##   userId movieId rating genres
##    <int>   <int>  <int>  <int>
## 1  69878   10677     10    797
```

Distribution of the ratings

The ratings can either be a full star rating or a half star rating. The scale ranges from 0.5 to 5 stars. The following histogram shows that full star ratings are given far more often than half star ratings. It also visualizes nicely that users tend to give higher ratings more often then lower ratings.

```
edx %>%
    mutate(group = ifelse(rating%%1 == 0, "whole star", "half star")) %>%
    ggplot(aes(rating, fill = group)) + geom_histogram(binwidth = 0.25,
    color = "black") + scale_y_continuous(breaks = c(seq(0, 3e+06,
    500000))) + ggtitle("Rating distribution")
```



Movies

The numbers of ratings varies greatly for individual films, with the most rated movie being *Pulp Fiction* (1994) with over 31,000 ratings and 126 movies that were rated only once. For the latter, predictions of future ratings will be difficult.

```
# Most rated films
edx %>%
    group_by(title) %>%
    summarize(n_ratings = n()) %>%
    arrange(desc(n_ratings))
```

```
## # A tibble: 10,676 x 2
##    title                                                n_ratings
##    <chr>                                                    <int>
##  1 Pulp Fiction (1994)                                      31362
##  2 Forrest Gump (1994)                                      31079
##  3 Silence of the Lambs, The (1991)                         30382
##  4 Jurassic Park (1993)                                     29360
##  5 Shawshank Redemption, The (1994)                         28015
```

```
##  6 Braveheart (1995)                                                  26212
##  7 Fugitive, The (1993)                                               25998
##  8 Terminator 2: Judgment Day (1991)                                  25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)       25672
## 10 Apollo 13 (1995)                                                   24284
## # ... with 10,666 more rows
```

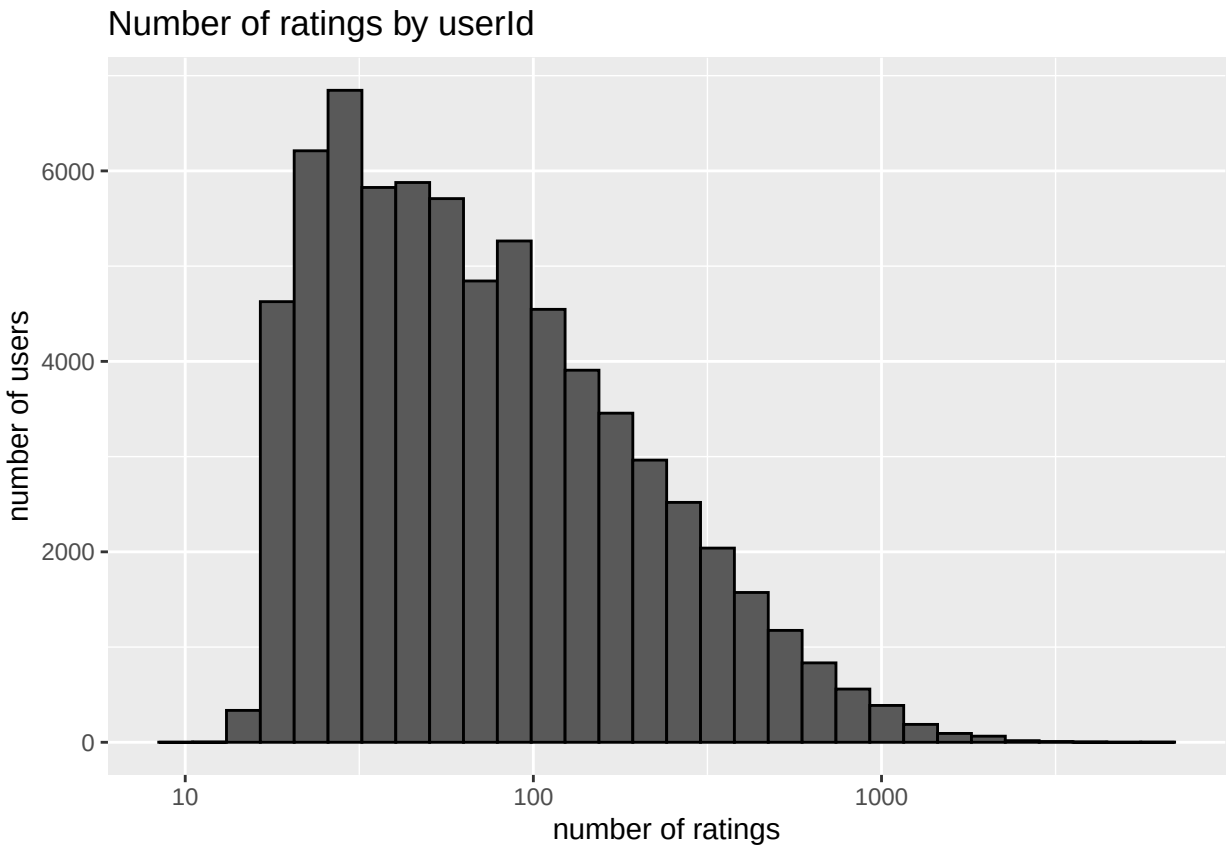
```
# Number of movies rated once
edx %>%
    group_by(title) %>%
    summarize(n_ratings = n()) %>%
    filter(n_ratings == 1) %>%
    summarise(n = n()) %>%
    pull()
```

```
## [1] 126
```

Users

As mentioned above, there are 69,878 individual users in the `edx` dataset. If every user rated every movie once we would have a dataset containing 746,087,406 observations in the entire dataset. We actually only have 9,000,055 observations - far less! This implies that not every user rated every movie or that different users rated the same movie, which is also the core of the Recommendation Systems problem. Each outcome has a different set of predictors. We can use a histogram to visualize the amount of movies users rated.

```
edx %>%
    group_by(userId) %>%
    summarize(count = n()) %>%
    ggplot(aes(count)) + geom_histogram(bins = 30, color = "black") +
    scale_x_log10() + ggtitle("Number of ratings by userId") +
    labs(x = "number of ratings", y = "number of users")
```



Since most data falls to the right, we have a right-skewed distribution. Different users rate different movies and a different number of movies. Most users have rated between 30 and 100 movies.

Timestamps

To investigate whether the time has an impact on the rating, we first calculate the duration during which movies were rated.

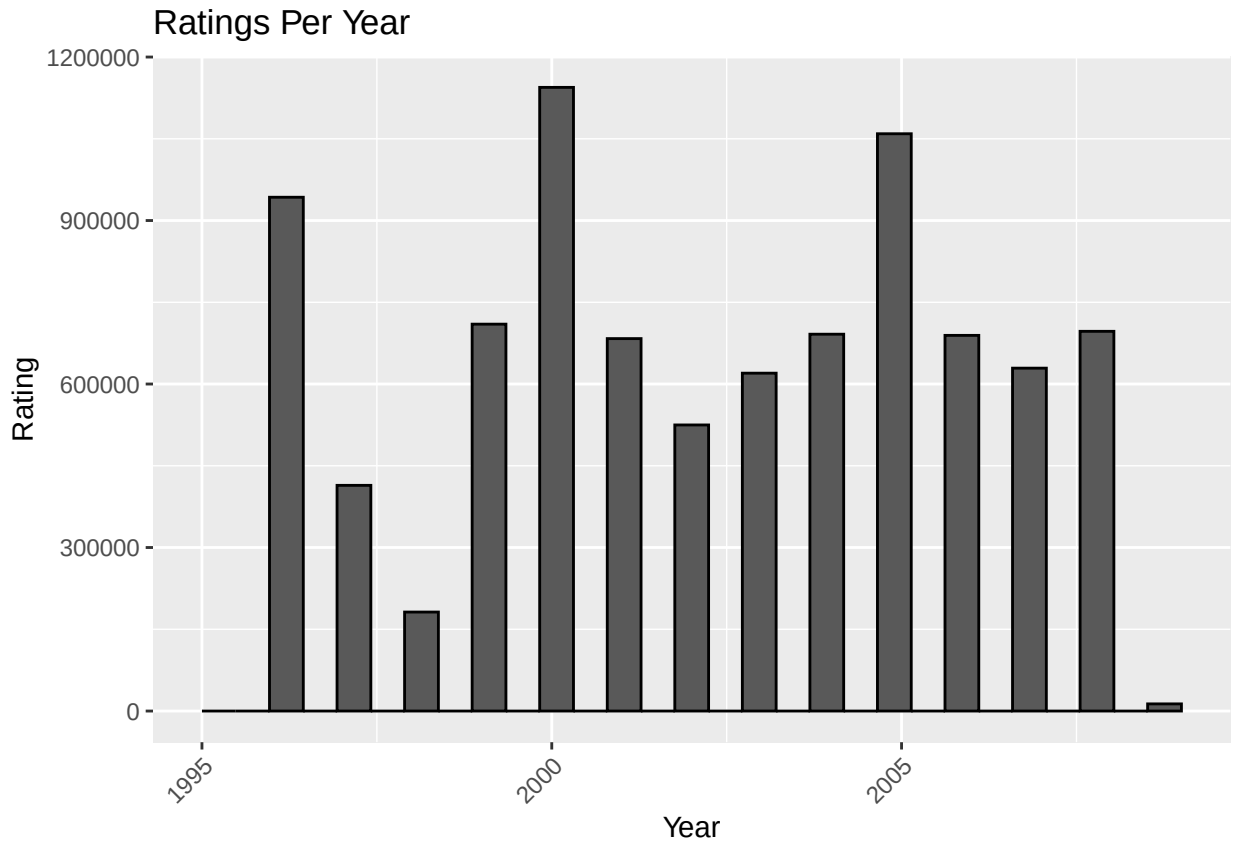
```
# first and last rating, duration
edx %>%
    mutate(min = date(as_datetime(min(edx$timestamp))), max =
     ↪  date(as_datetime(max(edx$timestamp))),
        duration = time_length(max - min, "years")) %>%
    select(min, max, duration) %>%
    distinct() %>%
    as_tibble()
```

```
## # A tibble: 1 x 3
##   min        max        duration
##   <date>     <date>        <dbl>
```

```
## 1 1995-01-09 2009-01-05     14.0
```

The edx dataset contains ratings from 14 years, from 1995 to 2009. We plot the absolute number of ratings for each year.

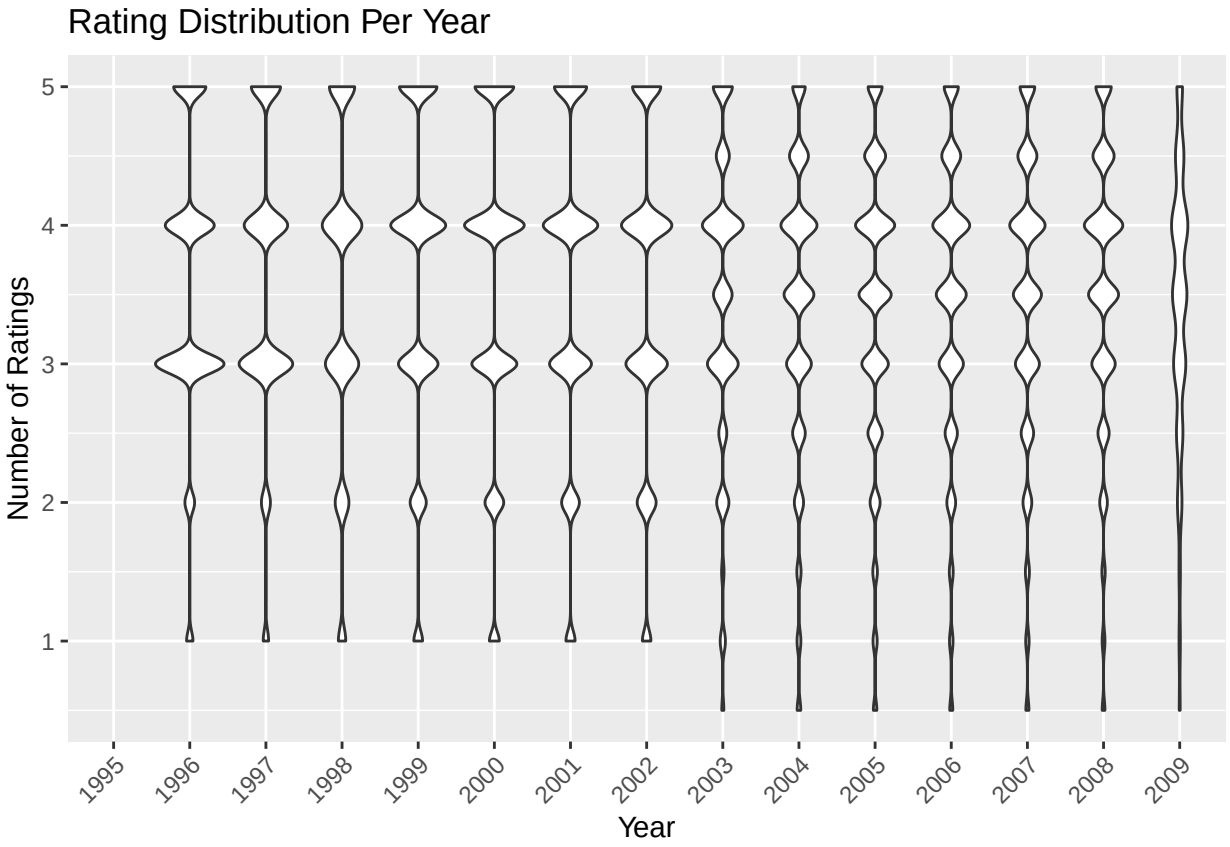
```
# total ratings per year
edx %>%
    mutate(year = year(as_datetime(timestamp, origin = "1970-01-01"))) %>%
    ggplot(aes(x = year)) + geom_histogram(color = "black") +
    ggtitle("Ratings Per Year") + xlab("Year") + ylab("Rating") +
    theme(plot.title = element_text(hjust = 0), plot.subtitle =
    ↪  element_text(hjust = 0.5),
       legend.position = "bottom", axis.text.y = element_text(hjust = 1),
       axis.text.x = element_text(hjust = 1, angle = 45))
```



Almost no movies were rated in both the first and the last year of the dataset with notably more ratings in the years 1996, 2000 and 2005. To get a better idea about the actual ratings that were given in each year, we use a violin plot which is usually used for continuous data.

```
# ratings per year, violin plot
edx_validation_set %>%
```

```
    mutate(year = year(as_datetime(timestamp, origin = "1970-01-01"))) %>%
    group_by(year) %>%
    mutate(year = as.factor(year)) %>%
    ggplot(aes(year, rating)) + geom_violin() + ggtitle("Rating Distribution
     ↪  Per Year") +
    xlab("Year") + ylab("Number of Ratings") + theme(plot.title =
     ↪  element_text(hjust = 0),
    plot.subtitle = element_text(hjust = 0.5), legend.position = "bottom",
    axis.text.y = element_text(hjust = 1), axis.text.x = element_text(hjust
     ↪  = 1,
        angle = 45))
```



Interestingly, it looks like half-star ratings were only given from 2003 onwards.

Genres

There are a total of 797 different genres or genre combinations as mentioned in Section Exploratory Data Analysis.

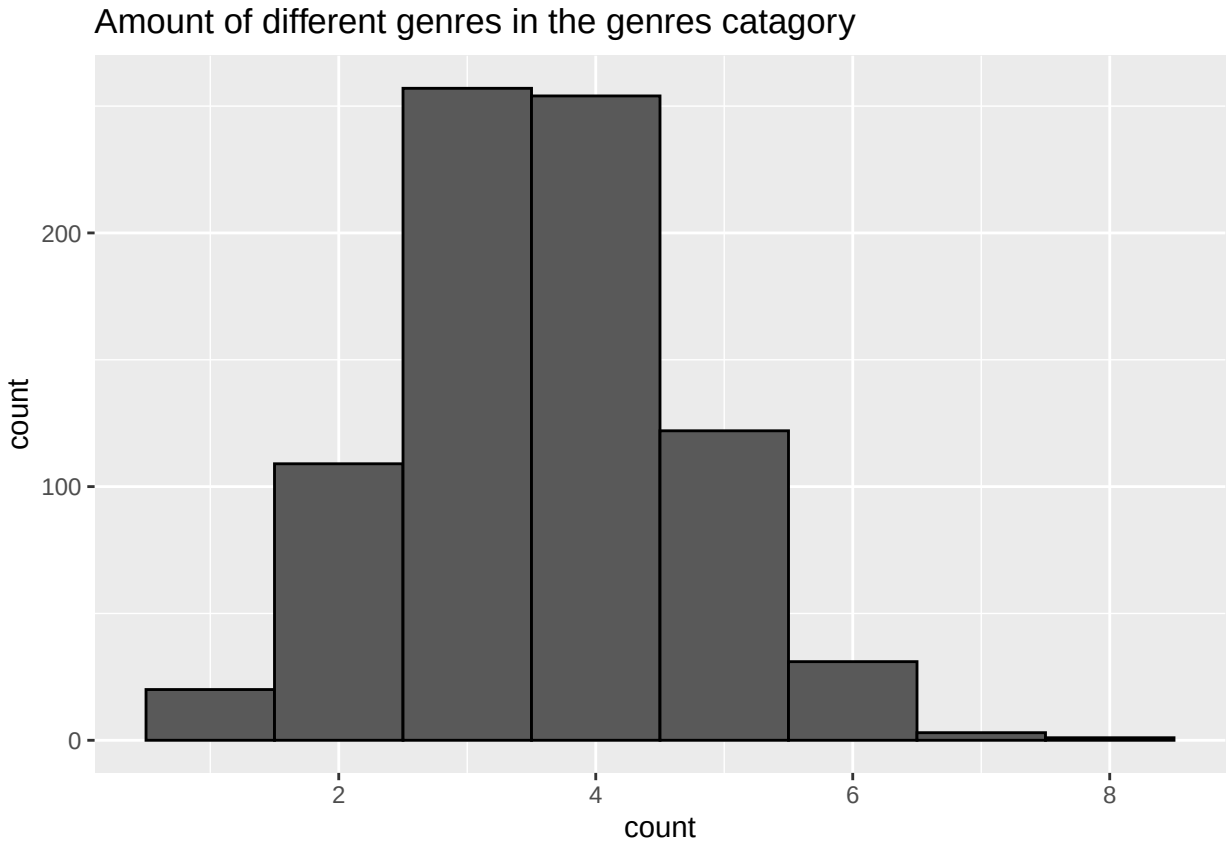
```r
# categories with the least and the most movies
genres <- tibble(count = str_count(edx$genres, fixed("|")), genres =
↪  edx$genres) %>%
    group_by(count, genres) %>%
    summarise(n = n()) %>%
    arrange(n)
```

```r
genres[c(1:5, (nrow(genres) - 5):nrow(genres)), 2:3] %>%
    knitr::kable()
```

genres	n
Adventure|Mystery	2
Documentary|Romance	2
Action|War|Western	2
Action|Animation|Comedy|Horror	2
Crime|Drama|Horror|Sci-Fi	2
Drama|Romance	259355
Comedy|Drama|Romance	261425
Comedy|Drama	323637
Comedy|Romance	365468
Comedy	700889
Drama	733296

Most movies are categorized by three or four different genres. The genres are approximately normally distributed.

```r
# how many genres counts how often
edx %>%
    group_by(genres) %>%
    summarize(n = n(), count = as.numeric(str_count(genres, fixed("|")) +
        1)) %>%
    distinct() %>%
    ggplot(aes(count)) + geom_histogram(bins = 8, color = "black") +
    ggtitle("Amount of different genres in the genres catagory")
```


Amount of different genres in the genres catagory
200
100
0
count
2
4
6
8
count

Method development

This machine learning challenge is to build a Recommendation Systems with given data to apply to unknown data. Because we are dealing with large data and our predictors are numeric, not categorical, we will use matrix algebra, regularization and finally matrix factorization to solve this machine learning problem. Because of the movie rating distribution we will apply regularization and a penalty term to the models in this project. Regularization is a technique used to reduce the error by fitting a function approximately on the given train set and avoid overfitting. Overfitting is 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 to deal with this problem by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values. To see how well our machine learning algorithm works and decide which model performs best, we have to define a *loss function*. It assigns to each model the "cost" caused by the decision and maps it onto one real number. We have to minimize the loss function in order to optimize the algorithm. We write now the loss function that computes the RMSE:

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

The actual rating for movie i by user u is denoted as $y_{u,i}$ and our prediction of the observation as $\hat{y}_{u,i}$. N is the number of user/movie combinations and the sum occurs over all these combinations. The written function to compute the RMSE for vectors of ratings and their corresponding predictions is:

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

Model 1: Avarage movie rating

The simplest model we can build is one in which we ignore both user and movie effects and just take the average with an error term. The respective formula looks like this:

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

where μ is the average rating across all entries and $\epsilon_{u,i}$ are the independent errors sampled from the same distribution centered at 0.

Model 2: Movie effect model

Since some movies are in general rated higher than others, we will now add an independent error term or bias b_i for movies. This term averages the rankings for each movie i minus the

predicted mean μ. The new model is:

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

Model 3: Movie and user effect model

Some users (with more than 100 ratings performed) rate movies generally far better than others. That means we have to consider a user bias. We therefore improve our model further:

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

where b_u is a user-specific effect. We compute b_u as the average of $y_{u,i} - \mu - b_i$.

Model 4: Regularized movie and user effect model

Sometimes movies were rated by very few users, in some cases even only once. In those cases we have more uncertainty resulting in larger absolute estimates for b_i. This results in overfitting or over-train of the model. Regularization achieves the same goal as confidence intervals when you are only able to predict a single number, not an interval. We apply regularization to reduce the effect outliers will have on our predictions and introduce an additional penalty term. We regularize both the movie bias term b_i and the user bias term b_u. The new model is:

$$\frac{1}{N}\sum_{u,i}(Y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2)$$

where the first term is the least squares equation and the last term is the penalty with bias terms. λ is a tuning parameter that must be selected to minimize the biases in our model. We use cross-validaton to tune λ.

Model 5: Matrix factorization

As mentioned earlier, the `edx` dataset is tidy and each observation is stored in one row. If we focus on the three most important variables of the dataset - userId, movieId and rating - we can present the problem differently. We can create a matrix with userId's as rows, movieId's as columns and the ratings as cell values.

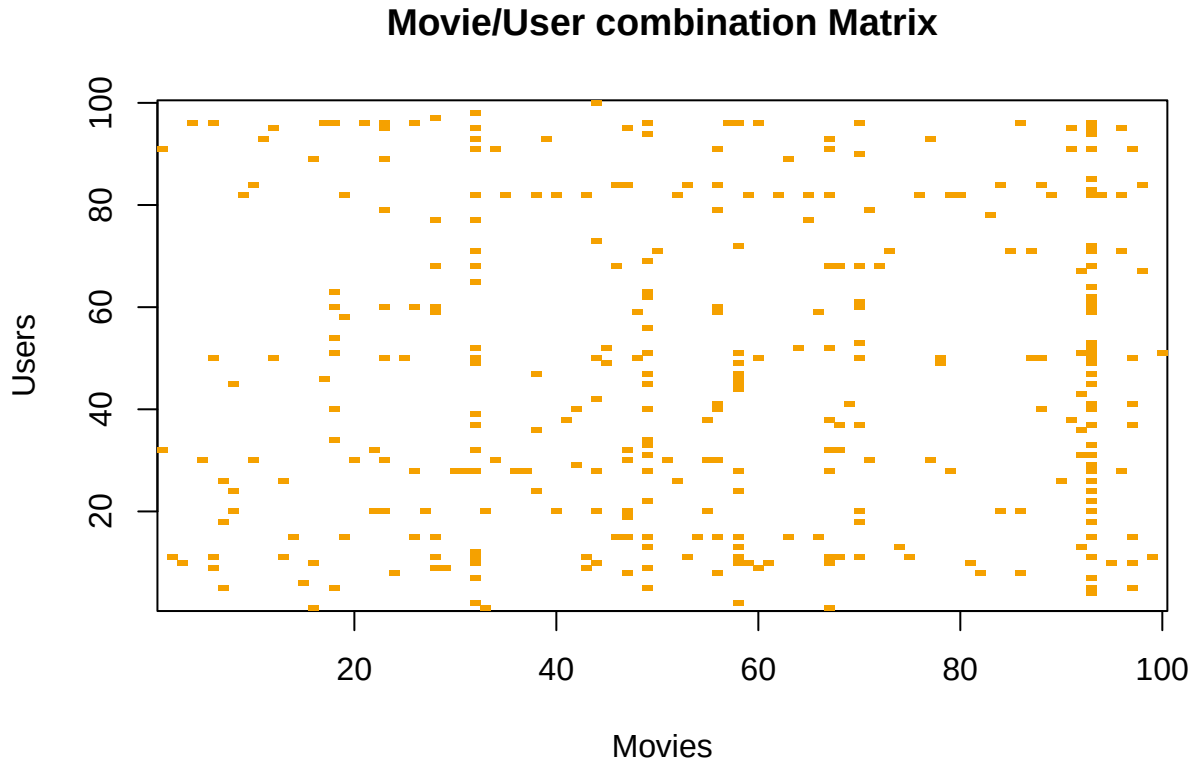
```
# recommendations system problem as a matrix
set.seed(1, sample.kind = "Rounding")
random_users <- sample(unique(edx$userId), 100)
user_movie_matrix <- edx %>%
    filter(userId %in% random_users) %>%
    select(userId, movieId, rating) %>%
    mutate(rating = 1) %>%
```

```
    pivot_wider(names_from = "movieId", values_from = "rating") %>%
    select(sample(ncol(.), 100)) %>%
    as.matrix() %>%
    t(.)
```

```
# plot the matrix
class(user_movie_matrix)
```

```
## [1] "matrix" "array"
```

```
user_movie_matrix %>%
    image(1:100, 1:100, ., xlab = "Movies", ylab = "Users")
title("Movie/User combination Matrix")
```




```
# rm(random_users)
```

Matrix factorization was used to win the Netflix challenge (Bell *et al.*, s.d.) and became widely known because of it. The concept is to divide the matrix with the ratings $R_{m \times n}$ into

two smaller matrices of lower dimension $P_{k\times m}$ and $Q_{k\times n}$, such that the product is

$$R \approx P'Q$$

Therefore, we need to transform our data into a matrix like the one above. We could transform the `edx` dataset into a matrix using tidyr's `pivot_wider` function like this:

However, this computation uses a huge amount of RAM and is therefore quite time consuming on a regular laptop, in this case on average about 2 seconds for an `edx` subset with only 3.000 instead of all 69.878 users. Also the following matrix factorization optimization is quite challenging, so instead we will use the `recosystem` package which uses parallel matrix factorization and has been built to specifically deal with Recommender System problems. The package can be found online (https://github.com/yixuan/recosystem) and its usage is explained here: https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html. It is an R wrapper of the LIBMF library (https://www.csie.ntu.edu.tw/~cjlin/libmf/), an open source library for Recommender System using parallel marix factorization.

Results

Model 1: Avarage movie rating

The starting point for our model development is simply to use the mean of all ratings as the expected $\hat{\mu}$. The formula looks like this:

$$\hat{y}_{u,i} = \hat{\mu} + \epsilon_{u,i}$$

We run the calculations with the following code:

```
### model 1: average across every entry, no user or movie
### effect calculate the average, then the RMSE
mu_hat <- mean(edx_train_set$rating)
model_1_rmse <- RMSE(edx_validation_set$rating, mu_hat)
```

```
print(mu_hat)
```

```
## [1] 3.5125
```

```
print(model_1_rmse)
```

```
## [1] 1.0601
```

The calculated value for $\hat{\mu}$ is 3.512 and the corresponding RMSE is 1.06005. In this particular case the RMSE is the standard deviation of the distribution of ratings.

Model 2: Movie effect model

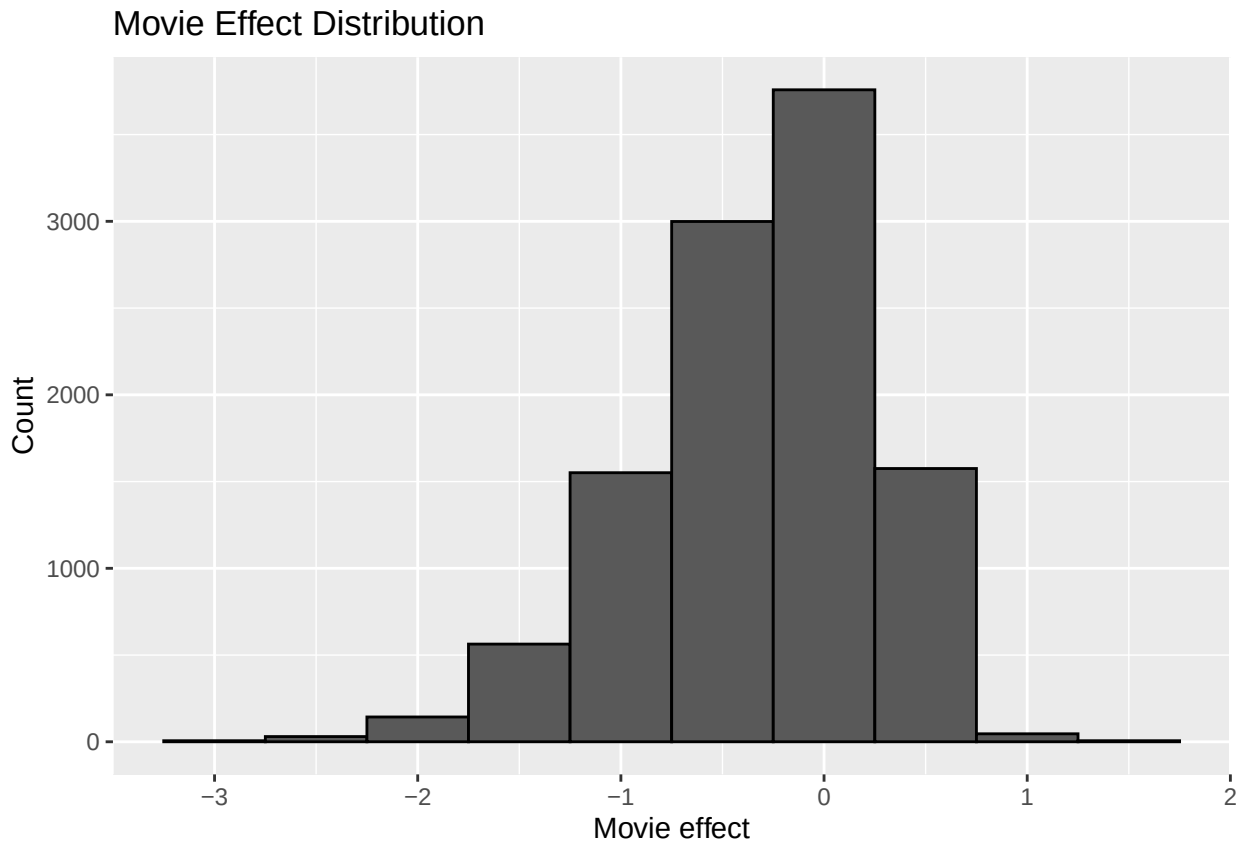
In the second model we add the bias term b_i for movies.

$$\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i + \epsilon_{u,i}$$

```
# calculate movie effect b_i
b_i <- edx_train_set %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu_hat))
```

We can see that these estimates are left skrewed.

```
# plot movie effect b_i
b_i %>%
    ggplot(aes(x = b_i)) + geom_histogram(bins = 10, color = "black") +
    ggtitle("Movie Effect Distribution") + xlab("Movie effect") +
    ylab("Count")
```




```
# predict all unknown ratings with mu and b_i
predicted_ratings <- edx_validation_set %>%
    left_join(b_i, by = "movieId") %>%
    mutate(pred = mu_hat + b_i) %>%
    pull(pred)
model_2_rmse <- RMSE(edx_validation_set$rating, predicted_ratings)
```

```
# calculate RMSE of movie ranking effect
print(model_2_rmse)
```

```
## [1] 0.94296
```

Including the movie bias improves the RMSE for this model to 0.94296.

Model 3: Movie and user effect model

When we include the user effect our model changes to:

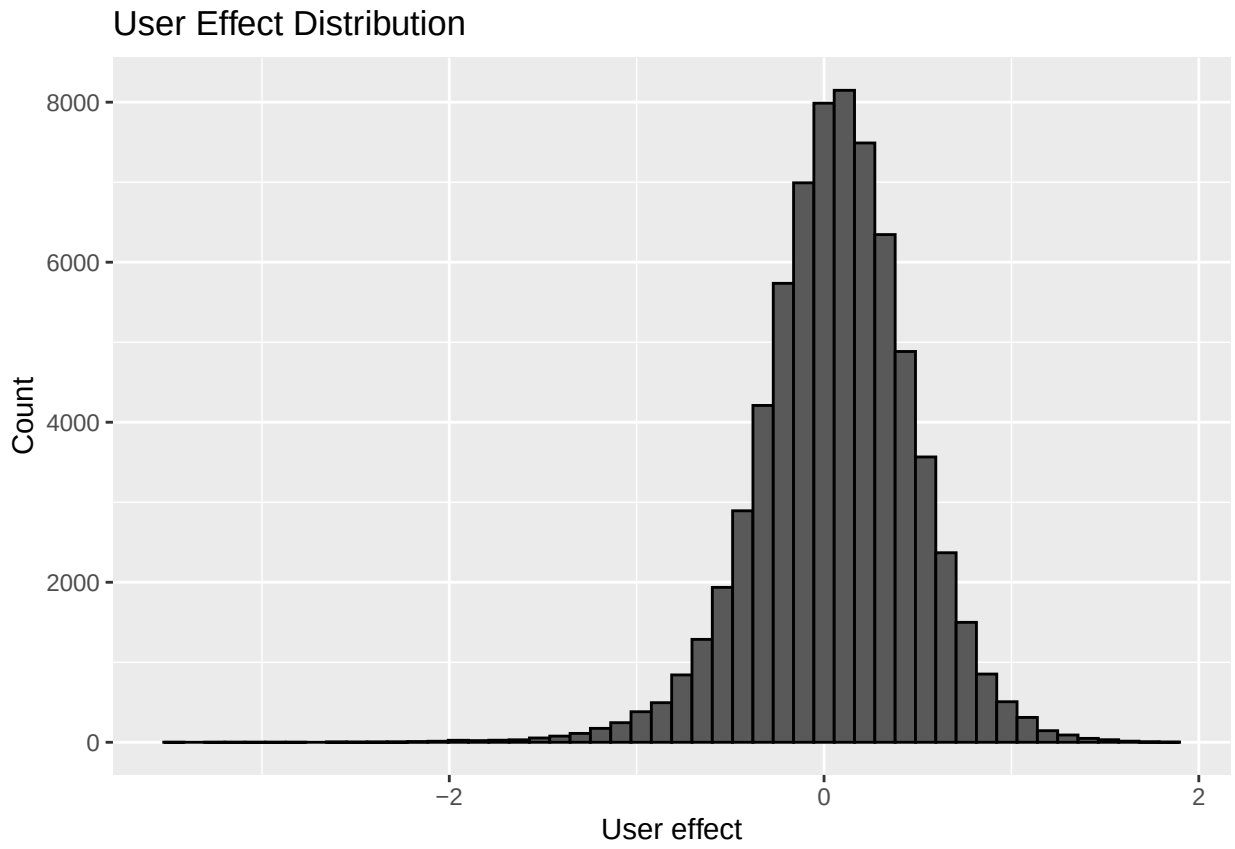
$$\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u + \epsilon_{u,i}$$

We calculate the user effect in a similar way to the movie effect.

```
# calculate user effect b_u
b_u <- edx_train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu_hat - b_i))
```

And plot it.

```
# plot movie effect b_i
b_u %>%
    filter() %>%
    ggplot(aes(x = b_u)) + geom_histogram(bins = 50, color = "black") +
    ggtitle("User Effect Distribution") + xlab("User effect") +
    ylab("Count")
```



The user effect is normally distributed. Next, we predict the ratings for the `edx_validation_set` and compute the RMSE:

```
# predict all unknown ratings with mu, b_i and b_u
predicted_ratings <- edx_validation_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu_hat + b_i + b_u) %>%
    pull(pred)
# calculate RMSE with movie and user effect
model_3_rmse <- RMSE(edx_validation_set$rating, predicted_ratings)
```

```
print(model_3_rmse)
```

```
## [1] 0.86468
```

The RMSE is now 0.86468, already far better than the fist model.

Model 4: Regularized movie and user effect model

As mentioned in the method development section, we now add a penalty term λ to the model. The new variable is a tuning parameter.

$$\frac{1}{N}\sum_{u,i}(Y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2)$$

We test values between 0 and 10 in steps of 0.25 for λ and plot the results. Therefore, we first define a fuction which computes the RMSE for a given set of `train_set`, `validation_set` and λ.

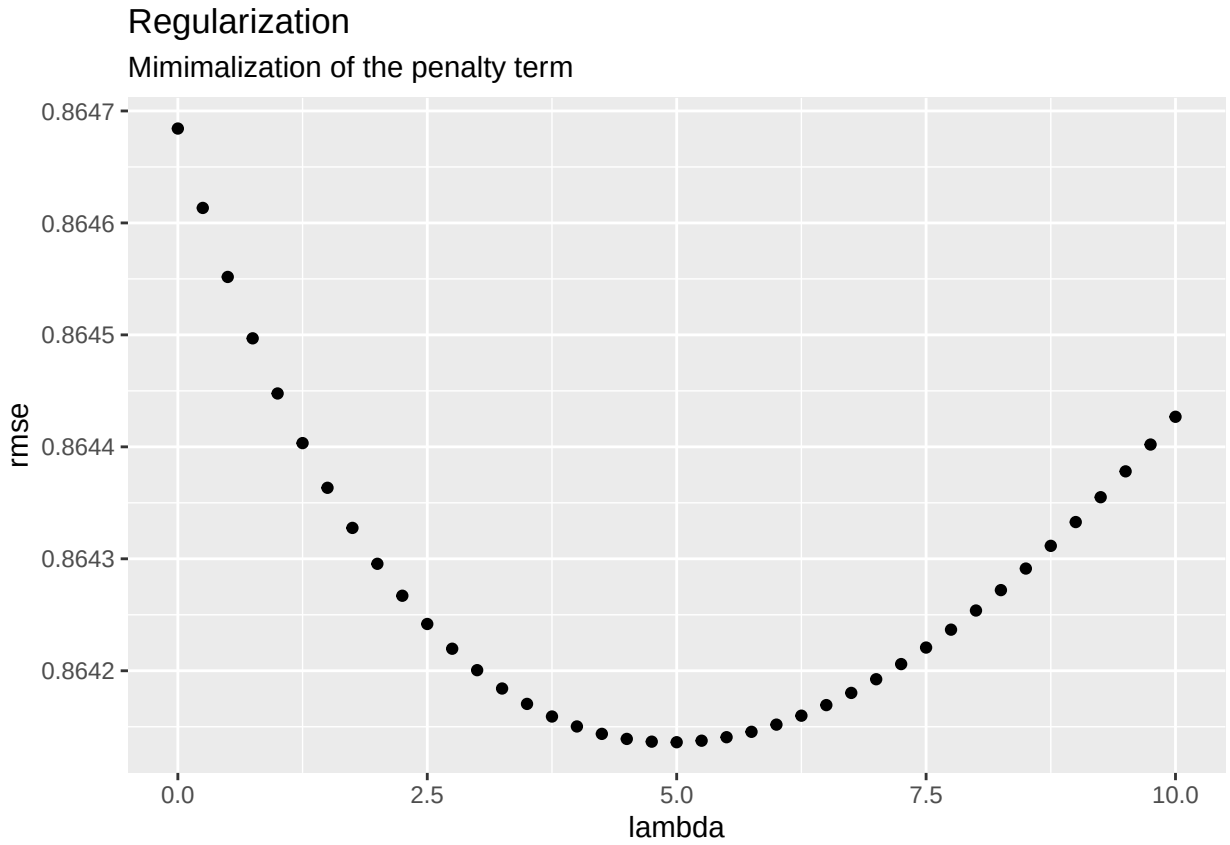
```
# write function to output and save RMSE of each lambda
regularization_model <- function(train_set, validation_set, lambda) {
    # calc. mu_hat (average rating across edx_traning_set)
    mu_hat <- mean(train_set$rating)
    # calc. regularized movie bias term
    b_i <- train_set %>%
        group_by(movieId) %>%
        summarize(b_i = sum(rating - mu_hat)/(n() + lambda))
    # calc. regularized user bias term
    b_u <- train_set %>%
        left_join(b_i, by = "movieId") %>%
        group_by(userId) %>%
```

```
        summarize(b_u = sum(rating - b_i - mu_hat)/(n() + lambda))
    # compute validation_set predictions
    predicted_ratings <- validation_set %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        filter(!is.na(b_i), !is.na(b_u)) %>%
        mutate(pred = mu_hat + b_i + b_u) %>%
        pull(pred)
    # output RMSE
    return(RMSE(predicted_ratings, validation_set$rating))
}
```

We use cross-validaton to tune λ, plot them against the resulting RMSEs and report the best result.

```
# write lambda sequence
lambdas <- seq(0, 10, 0.25)
# sapply lambdas with edx_train_set and edx_validation_set
rmses <- sapply(lambdas, regularization_model, train_set = edx_train_set,
    validation_set = edx_validation_set)
```

```
# plot lambdas vs. rmses
tibble(lambda = lambdas, rmse = rmses) %>%
    ggplot(aes(lambda, rmse)) + geom_point() + ggtitle("Regularization",
    subtitle = "Mimimalization of the penalty term")
```




```
# for which lambda is RMSE minimized
lambdas[which.min(rmses)]
```

```
## [1] 5
```

```
# save tuned lambda
tuned_lambda <- lambdas[which.min(rmses)]
# save rsme for model 4
model_4_rmse <- min(rmses)
print(model_4_rmse)
```

```
## [1] 0.86414
```

The RMSE for this model with λ set to 5.00 is 0.86414 and thus fulfills the criterion of being smaller than 0.86490. But can we go even further?

Model 5: Matrix factorization

The concept of matrix factorization has been described in an earlier section. The package vignette of the 'recosystem' package discribes how to use it (https://cran.r-project.org/

web/packages/recosystem/vignettes/introduction.html). Once we created a model object and tuned it, we predict the ratings for the 'validation' data and calculate the RMSE.

```
# can't use tibble or data_frames because it uses to much
# RAM use recosystem package
set.seed(9999, sample.kind = "Rounding")

# convert data into recosystem format
edx_train_data <- with(edx_train_set, data_memory(user_index = userId,
    item_index = movieId, rating = rating))
edx_validation_data <- with(edx_validation_set, data_memory(user_index =
↪  userId,
    item_index = movieId, rating = rating))

# create a recommendation model object
r <- Reco()

# select the best tuning parameters
opts <- r$tune(edx_train_data, opts = list(dim = c(10, 20, 30),
    lrate = c(0.1, 0.2), costp_l2 = c(0.01, 0.1), costq_l2 = c(0.01,
        0.1), nthread = 4, niter = 10))
# train the model
r$train(edx_train_data, opts = c(opts$min, nthread = 4, niter = 20))

# predict ratings
predicted_ratings <- r$predict(edx_validation_data, out_memory())

# calculate rmse
model_5_rmse <- RMSE(edx_validation_set$rating, predicted_ratings)
```

```
print(model_5_rmse)
```

```
## [1] 0.78552
```

Model comparison

```
rmse_results <- tibble(method = c("Just the average", "Movie effect",
    "Movie and user effect", "Regulized movie and user effect",
    "Matrix factorization"), RMSE = c(model_1_rmse, model_2_rmse,
    model_3_rmse, model_4_rmse, model_5_rmse))
knitr::kable(rmse_results, digits = 5)
```

method	RMSE
Just the average	1.06005
Movie effect	0.94296
Movie and user effect	0.86468
Regulized movie and user effect	0.86414
Matrix factorization	0.78552

We can see that the linear model improves with each additional bias term and with the regularization. Although the regulized linear model already fulfills the criterion (RMSE $<$ 0.86490), the RMSE with matrix factorization is even 9.1 % better and will therefore be used for the final test.

Final Test using the `final_holdout_test`

We verified our algorithm choice using the training and validation data. Now that we found the optimal model on the 'edx_validation_set', training is repeated with the entire 'edx' dataset. More data reduces variability of the model and should improve the overall prediction. Finally, we evaluate the the newly trained model on the 'final_holdout_test' set. The final RMSE is computed as follows:

```
# convert data into recosystem format
edx_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 a recommendation model object
r_final <- Reco()

# select the best tuning parameters
opts_final <- r$tune(edx_data, opts = list(dim = c(10, 20, 30),
    lrate = c(0.1, 0.2), costp_l2 = c(0.01, 0.1), costq_l2 = c(0.01,
        0.1), nthread = 4, niter = 10))
# train the model
r_final$train(edx_data, opts = c(opts_final$min, nthread = 4,
    niter = 20))

# predict ratings
predicted_ratings <- r$predict(final_holdout_test_data, out_memory())

# calculate rmse
matrix_fact_final_rmse <- RMSE(final_holdout_test$rating, predicted_ratings)
```

```
print(matrix_fact_final_rmse)
```

```
## [1] 0.78611
```

The final RMSE with matrix factorization is 0.78552. It is a bit higher then the RMSE we previously reported for the `edx_validation_set`, but this is very common for machine learning. This RMSE fulfills the specified criterion.

Conclusion

Brief summary

In this project we downloaded the R `MovieLens` dataset from an online source, then split it into three parts for training (81 %), validation (9 %) and a final test of the models performance (10 %). Next, we explored the data an visualized the effect that different parameters have on the movie rating. We used these gained insights to improve our model step by step. First, we created a linear model which is just the mean of the observed ratings. Based on this, we added bias terms for movie effect and user effect. Because we do not want the model to be over-trained, we added a penalty term for movies and users with a low number of ratings. The developed model has an RMSE of 0.86414, successfully falling below the target value (RMSE < 0.86490). We then approached the problem differently and used the `recosystem` package that uses the open-source LIBMF data base and received a RMSE of 0.78552. Finally, we retrained this model with the entire `edx` dataset and tested it on the `final_holdout_test` set to give an objective evaluation of the final model. We achieved an RMSE of 0.78611 which fulfills the specified criterion.

Limitations

We only used two predictors - userId and movieId - for both the linear and the matrix factorization model. We showed during exploratory data analysis that both the timestamps and the genres did also have an impact on the rating. State-of-the-art machine learning models should be able to consider these additional predictors to make even better predictions. Another flaw is that neither model is able to make any movie rating predictions for new users.

Future work

This report describes two different approaches to solve the `MovieLens` Recommendation System problem. These approaches could be further improved if k-fold cross-validation would be used to reduce their variability. It's also possible to approach this problem using content-based (Breese *et al.*, 2013) or collaborative filtering (Charu, 2016) which are widely used for Recommender Systems. Another possibility is to combine multiple results which was done by the winning team of the Netflix challenge who merged 107 different algorithmic approaches (Bell *et al.*, s.d.).

References

Bell R.M., Koren Y. & Volinsky C. (s.d.). The BellKor solution to the Netflix Prize kernel description. https://web.archive.org/web/20120304173001/http://www.netflixprize.com/assets/ProgressPrize2007_KorBell.pdf.

Breese J.S., Heckerman D. & Kadie C. (2013). Empirical analysis of predictive algorithms for collaborative filtering. arXiv preprint arXiv:1301.7363.

Charu C.A. (2016). Recommender systems: The textbook.

Irizarry R.A. (2019). Introduction to data science: Data analysis and prediction algorithms with r. CRC Press.

Lohr S. (2009). https://www.nytimes.com/2009/09/22/technology/internet/22netflix.html.

Pierson L. (2021). Data science for dummies. John Wiley & Sons.