Movie recommendation system: Predicting movie ratings in the MovieLens data set

Davi Alves Oliveira

Introduction

This report describes the development of an algorithm, implemented in the R language (R Core Team 2018), to predict ratings using a subset of the MovieLens data set. This subset is referred here as the edx data set and contains 9.000.055 ratings of 10677 movies by 69878 users, with six columns, namely 'rating', 'userId', 'movieId', 'timestamp', 'title' and 'genres'. In this analysis, 'rating' is the outcome and the other columns are used either as predictors or to generate new columns that are used as predictors. A detailed description of each column and columns derived from these six original ones is given in the next section (Method).

The code below, provided as part of the course material in the edx platform, was used to create the edx data set.

```
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes

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

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

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

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

movies <- str_split_fixed(
  readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(levels(movieId))[movieId],
         title = as.character(title),
         genres = as.character(genres))
```

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

# Validation set will be 10% of MovieLens data
set.seed(1) # set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)`. Otherwise, use
# `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,]

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

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

# Remove unecessary data from memory
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

The code below, in turn, sets the initial configurations, installing and/or loading the necessary additional packages and setting the theme of the plots.

```
# Install and/or load the necessary packages
# Note: this may take several minutes
if (!require(lubridate))
  install.packages("lubridate", repos = "http://cran.us.r-project.org")
if (!require(magrittr))
  install.packages("magrittr", repos = "http://cran.us.r-project.org")
if (!require(ggrepel))
  install.packages("ggrepel", repos = "http://cran.us.r-project.org")
if (!require(ggthemes))
  install.packages("ggthemes", repos = "http://cran.us.r-project.org")

# Set the theme for the plots
theme_set(theme() + theme_hc())
```

The code below extracts the basic information from the data set.

```
# Number of ratings
n_ratings = nrow(edx)

# Number of movies
n_movies = edx %>%
  group_by(movieId) %>%
  sample_n(1) %>%
  ungroup() %>%
  summarize(n_movies = n())

# Number of users
n_users = edx %>%
```

```
  group_by(userId) %>%
  sample_n(1) %>%
  ungroup() %>%
  summarize(n_users = n())

# Creates a tibble with the values for better visualization
tibble(n_ratings, n_movies, n_users)
```

```
## # A tibble: 1 x 3
##   n_ratings n_movies n_users
##       <int>    <int>   <int>
## 1   9000055    10677   69878
```

Columns

```
# Print the data frame columns' names
names(edx)
```

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

The goal of the algorithm is to predict the ratings of different movies by different users with a desired Root Mean Square Error (RMSE) smaller than 0.86490. To do this, the original data was pre-processed and analyzed following the instructions available in Irizarry (2019, sec. 34.7). The initial idea was to compare the results of this first type of models with results from other algorithms, such as KNN (K-Nearest Neighbors), but because of the size of the data set, the use of the train function of the caret package (Kuhn 2020) with algorithms such KNN resulted in memory allocation errors. Thus, only the approach described in Irizarry was followed. The packages used in this work were the tidyverse (Wickham et al. 2019), lubridate (Grolemund and Wickham 2011), magrittr (Bache and Wickham 2014), caret (Kuhn 2020), ggthemes (Arnold 2019), ggrepel (Slowikowski 2020) and beepr (Bååth 2018, the package beepr was used during the development and testing of codes by the use of the "beepr::beep()" function).

This report is organized in three additional sections: the Method section describes the steps of data cleaning, data exploration and modeling, the Results section describes the analysis of the final model and its validation, and the Conclusion section brings a summary of the results with limitations and suggestions for future work.

```
# Remove unecessary data from memory
rm(n_movies, n_users, n_ratings)
```

Method

The method consisted in one stage of data cleaning, one stage of data exploration and one stage of modeling. Data cleaning consisted in the pre-processing of the data, including the creation of new columns. Data exploration consisted in the visual inspection of the data and exclusion of columns considered unnecessary based this inspection. Finally, modeling consisted in the creation of linear models. Each stage is explained in the subsequent sections, but first a detailed description of the columns is given.

The columns 'userId' and 'movieId', as the names suggest, contains unique numerical identifications of each user and movie, respectively. These columns were used as originally found in the edx data set. The column 'title', with the titles and year of release of the movies, was split in two columns, 'title' and 'year'. The column 'timestamp', with the timestamp of the rating, was used in the creation of five additional columns that enabled better exploratory analysis. From the timestamps, the columns 'rating_year', 'rating_month', 'rating_day', 'rating_wday' and 'rating_hour' were created, containing, respectively, the year of the rating, the month of the rating, the day of the month of the rating (1 to 31), the day of the week of the rating (1 to 7), and the hour of the rating (0 to 23). The idea was to explore whether the moment of the rating, with different degrees of specificities, should be used as predictor of rating. Additionally, the column 'years_since_release' was created by subtracting the year of release from the year of the rating, thus representing how long the

movie was available for rating. Finally, the genres column was used to create columns of binary variables measuring whether or not the movie is of a certain genre. For example, the value of the column 'is_Action' is 1 if 'Action' in one of the genres in the 'genres' column and 0 otherwise. These columns were originally meant to be used as predictors, but due to memory allocation errors, described later, the columns were only used to compute the number of genres and then discarded.

Data cleaning

The following code was used to create the additional columns from the timestamps.

```
# Extract the year of the movie from the column 'title'. The pattern matches
# four numbers inside parenthesis and the capture group isolates only the
# numbers
edx = edx %>%
  mutate(
    year = str_match(title, '\\((\\d{4})\\)')[,2] %>% as.numeric(),
    time = as_datetime(timestamp),
    rating_year = year(time),
    rating_month = month(time),
    rating_day = day(time),
    rating_wday = wday(time),
    rating_hour = hour(time),
    years_since_release = rating_year - year)
```

From the 'genres' column, first a list of all the genres was created and stored in the vector 'genres'. Then, for each genre, a column was created with a binary variable representing whether a given movie is of a particular genre or not.

List of genres used in the edx data set

```
# Split the genres into individual genres, using the separator '|'.
# Compact the vector so it keeps only unique values
genres = edx %$%
  genres %>% str_split('\\|') %>% unlist() %>% unique()
genres
```

```
##  [1] "Comedy"             "Romance"            "Action"
##  [4] "Crime"              "Thriller"           "Drama"
##  [7] "Sci-Fi"             "Adventure"          "Children"
## [10] "Fantasy"            "War"                "Animation"
## [13] "Musical"            "Western"            "Mystery"
## [16] "Film-Noir"          "Horror"             "Documentary"
## [19] "IMAX"               "(no genres listed)"
```

The vector 'genres' was used to create the is_[genre] columns (e. g. is_Action, is_Comedy, etc), using the code below.

```
# Iterate through the values of the 'genres' vector.
# For each value, attach the 'is_' preffix and then create a column with this
# name, attributing the value 1 if the genre is present in the 'genres' column
# or 0 otherwise.
# The '!!variable :=' notation is used so that the value of the variable is
# attributed as the name of the column. See
# https://dplyr.tidyverse.org/articles/programming.html
for (g in genres) {
  genre = str_glue('is_', g)
```

```
  edx = edx %>%
    mutate(!!genre := ifelse(str_detect(.$genres, g), 1, 0))
}
```

After creating the new columns, the code below was used to analyze the distribution of genres. As can be observed, the most frequent genres are Drama and Comedy. Seven movies have no genre listed and 8.181 movies are listed as 'IMAX', which is not a genre per se, but format, which might significantly affect rating.

```
# Select the columns whose names starts with 'is_' (e. g. is_Action, is_Comedy)
# and show their sums, which are the number of movies in each genre
edx %>%
  select(starts_with('is_')) %>%
  colSums() %>%
  sort(decreasing = T)
```

```
##              is_Drama             is_Comedy             is_Action
##               3910127               3540930               2560545
##           is_Thriller          is_Adventure            is_Romance
##               2325899               1908892               1712100
##             is_Sci-Fi              is_Crime            is_Fantasy
##               1341183               1327715                925637
##           is_Children             is_Horror            is_Mystery
##                737994                691485                568332
##                is_War           is_Animation             is_Musical
##                511147                467168                433080
##            is_Western           is_Film-Noir         is_Documentary
##                189394                118541                 93066
##               is_IMAX is_(no genres listed)
##                 8181                     7
```

Data exploration

Data visualization was used to explore possible effects of the variables on rating. Special attention was given in this stage to the time-related variables, since it was not clear if the day of the week or the day of the month, for example, would have any effect on rating.

First, a column with the number of ratings per movie was created ('n_ratings'). The visual exploration consisted in plotting rating against each one of the time-related variables in a heat map with color representing the number of ratings, so possible interactions between these variables could be analyzed. Then, heat maps were generated for each variable of interest, as shown in sequence. The variables 'userId' and 'movieId' were not included in the exploratory analysis because the effects of these variables are already known.

```
# Count the number of ratings of each movie and store it in the 'n_ratings'
# column
edx = edx %>%
  group_by(movieId) %>%
  mutate(n_ratings = n()) %>%
  ungroup()
```

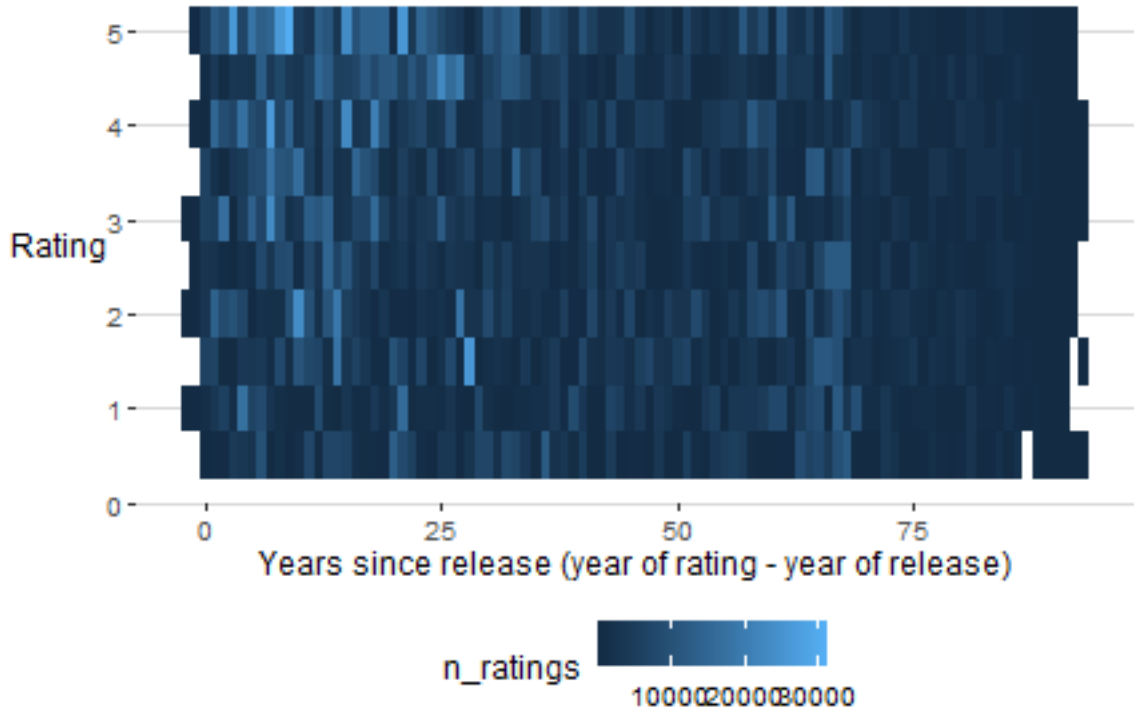
Years since release

It is already known that older movies tend to be more rated because of the longer time they are available for rating. Movies with more ratings also tend to be better rated (see Irizarry 2019, sec. 33.8). To visualize if these trends are also observed in the edx data set, a heat map was created with the code below. The right portion of the plot is darker than the left portion, which shows that older movies indeed are rated more

frequently, as expected considering the first trend. However, if the trend of movies with more ratings being better rated is present in the data set, it can not be observed with this plot, since there seems to be no clear pattern differing the top portion of the plot from the bottom portion.

Years since release

```
# Create a heatmap of rating versus years sinc release with fill representing
# number of ratings
# Note: plotting may take several minutes
edx %>%
  ggplot(aes(years_since_release, rating, fill = n_ratings)) +
    geom_tile() +
    ylab('Rating') +
    xlab('Years since release (year of rating - year of release)')
```



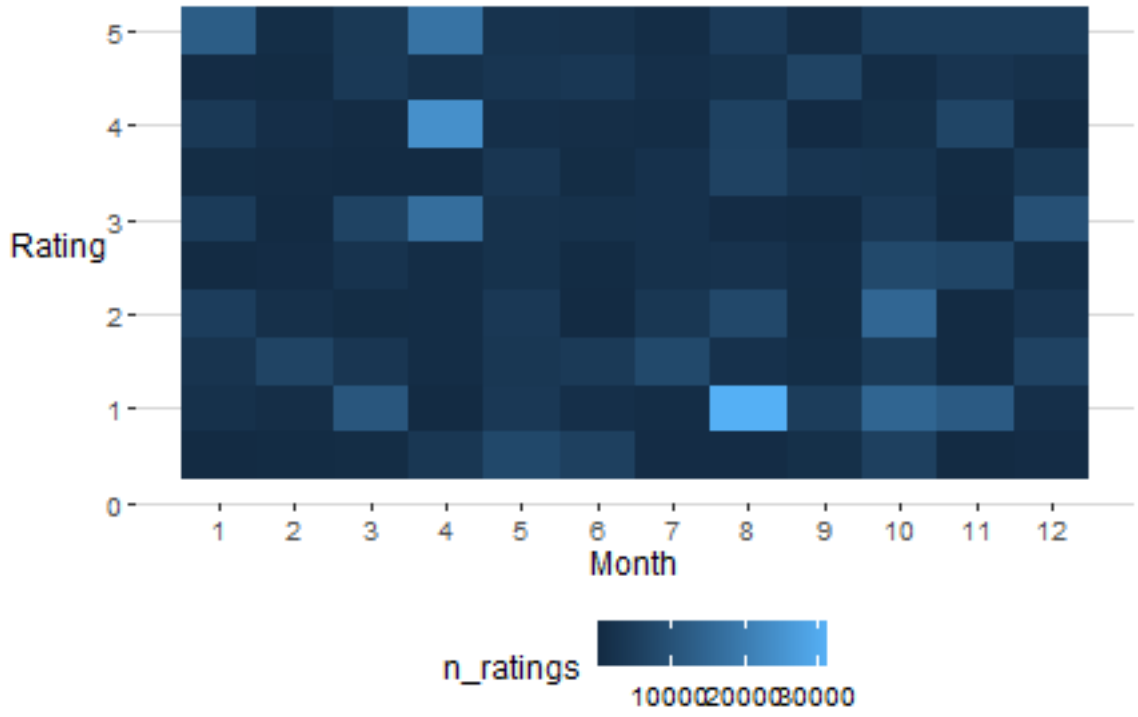
Month, day of the month, day of the week and hour of rating

The next four plots were generated following the same approach as the previous one with the variables month of rating, day of the month, day of the week and hour of rating. In summary: concerning months, some anomalies can be observed in April (4) and August (8), but with no overall clear linear pattern; concerning day of the month, no overall linear pattern is observed; concerning day of week, it can be observed that there are more ratings on Sundays and at the beginning of the week movies tend to be rated lower and; concerning hour of the day, no clear overall linear pattern can be observed.

Month of rating

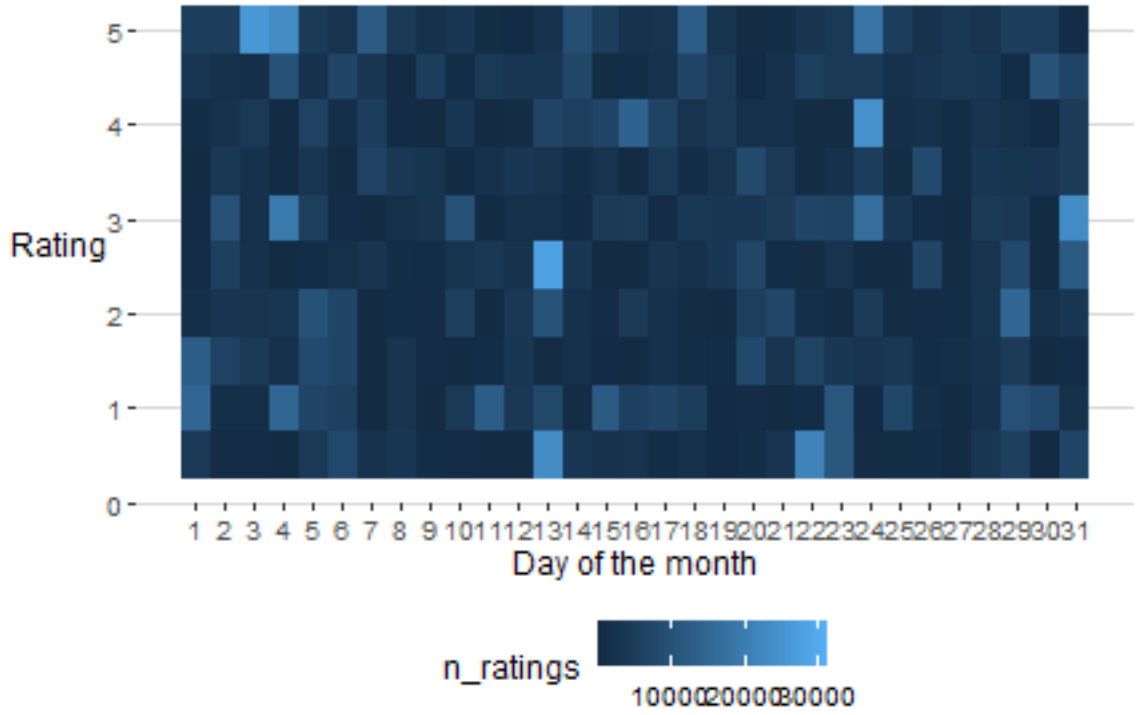
```
# Plot rating against month of rating, with fill representing the number of
# ratings
edx %>%
  ggplot(aes(rating_month, rating, fill = n_ratings)) +
    geom_tile() +
```

```
    scale_x_continuous(breaks = 1:12) +
    ylab('Rating') +
    xlab('Month')
```



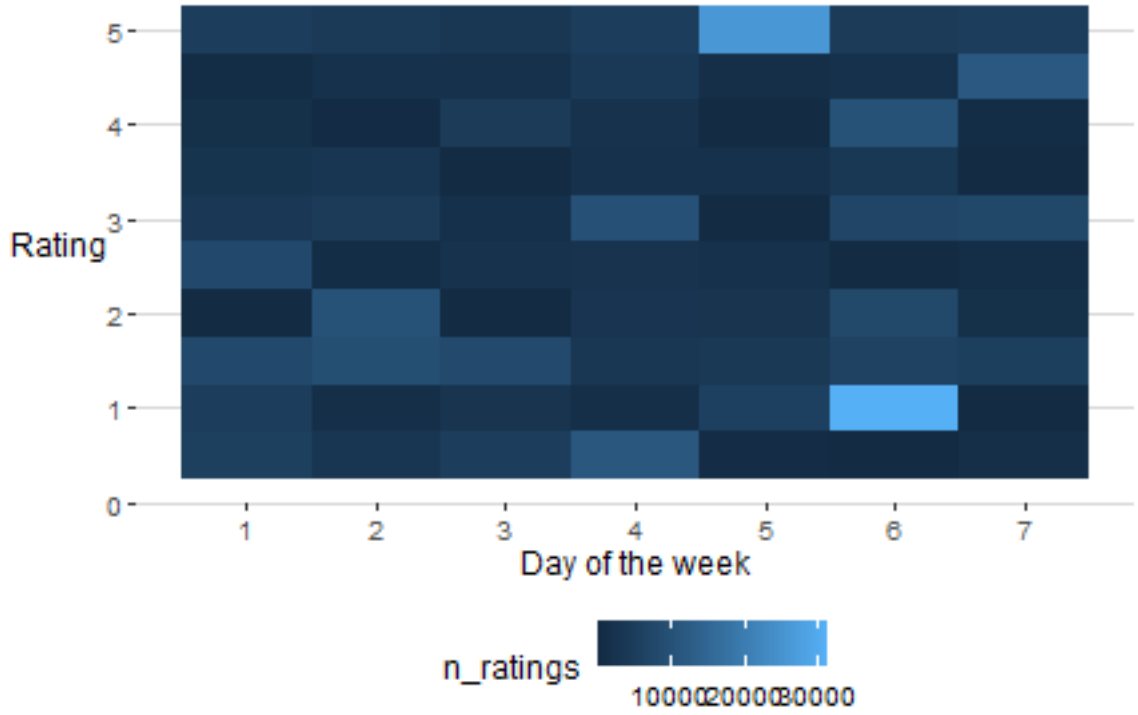
Day of the month

```
# Plot rating against day of the month, with fill representing the number of
# ratings
edx %>%
  ggplot(aes(rating_day, rating, fill = n_ratings)) +
    geom_tile() +
    scale_x_continuous(breaks = 1:31) +
    ylab('Rating') +
    xlab('Day of the month')
```



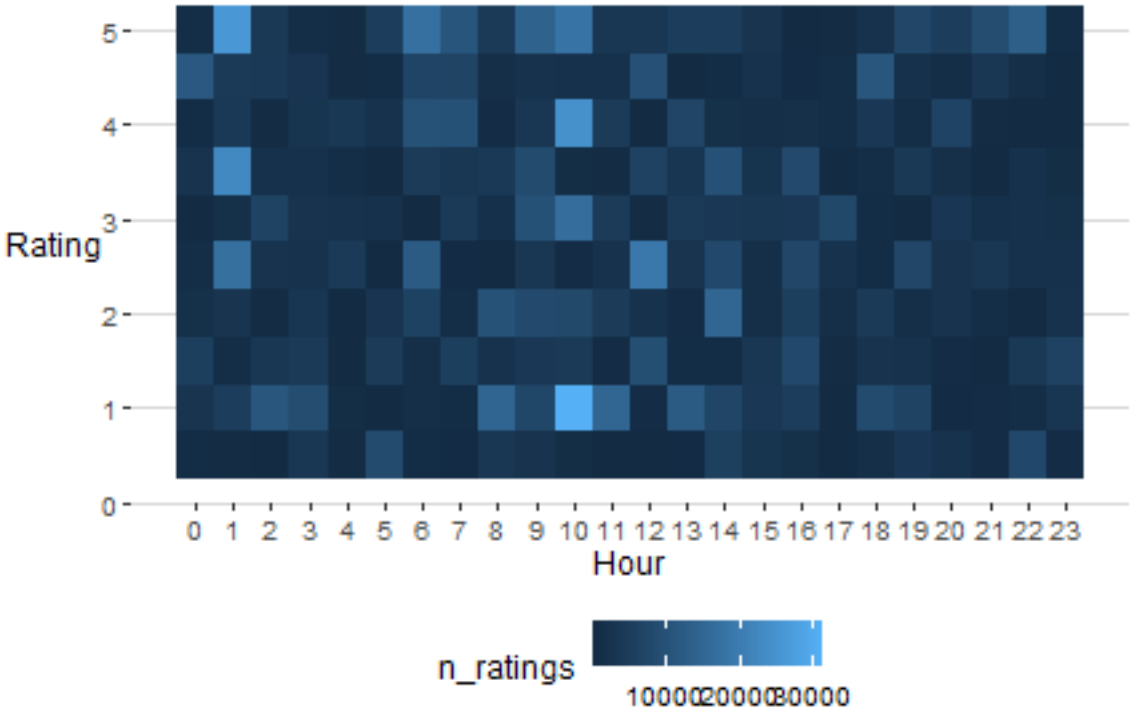
Day of the week

```
# Plot rating against day of the week, with fill representing the number of
# ratings
edx %>%
  ggplot(aes(rating_wday, rating, fill = n_ratings)) +
    geom_tile() +
    scale_x_continuous(breaks = 1:7) +
    ylab('Rating') +
    xlab('Day of the week')
```



Hour of rating

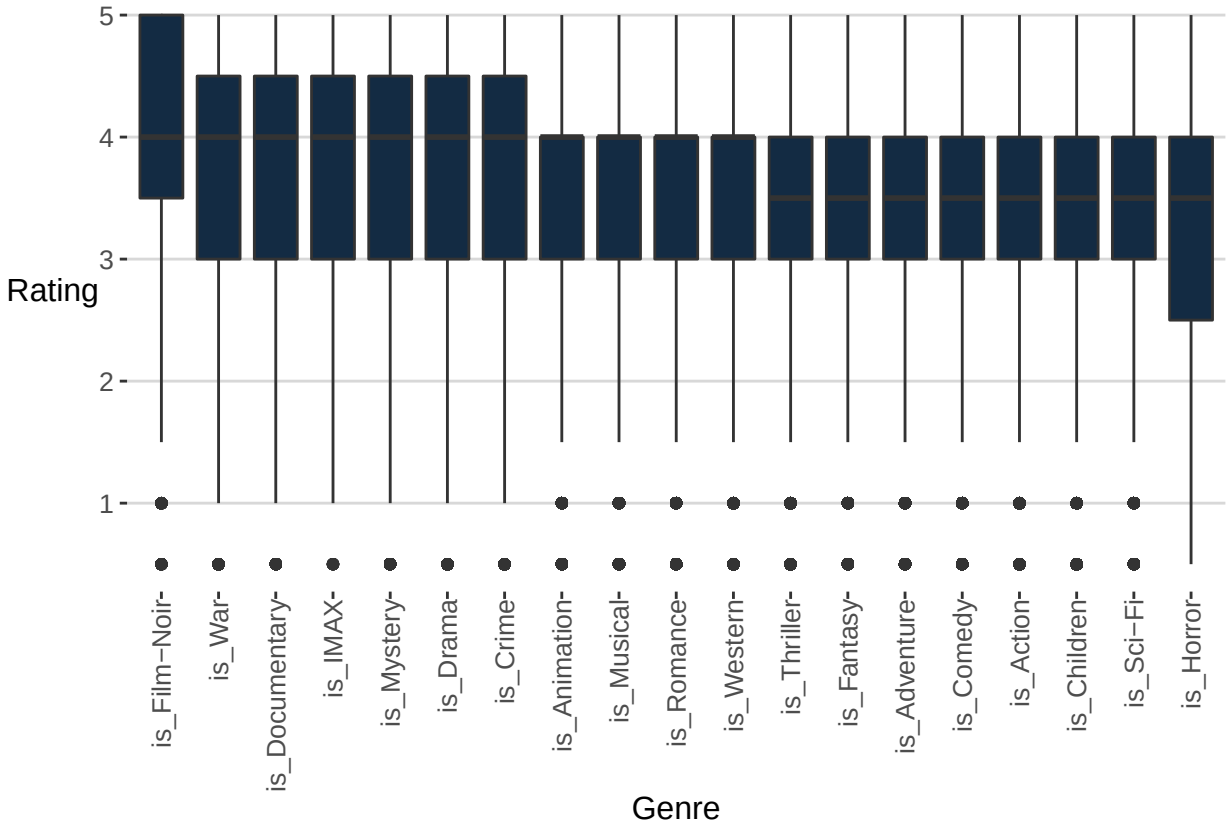
```
# Plot rating against hour, with fill representing the number of ratings
edx %>%
  ggplot(aes(rating_hour, rating, fill = n_ratings)) +
    geom_tile() +
    scale_x_continuous(breaks = 0:23) +
    ylab('Rating') +
    xlab('Hour')
```



Genres

To check a possible effect of genre on rating, a box plot of rating against genre was analyzed. Initially the idea was to analyze the whole data set, but the machine used in the analysis could not handle the task due to lack of memory. With the original edx data set, the code below returns an error message. Thus, after trials with increasingly smaller partitions, the code could run with 10% of the data set. It seems that genre has some effect on rating, with Film-Noir being better rated than Horror, for example.

```
# Set the seed
set.seed(2020)

# Generate indexes for partition
partIndex = createDataPartition(
  y = edx$rating, times = 1, p = 0.1, list = FALSE) %>% .[,1] %>% as_vector()

# Create subset of data and used it to generate a box plot of rating against
# genre
edx[partIndex, ] %>%
  select(rating, starts_with('is_')) %>%
  pivot_longer(-rating, names_to = 'genre', values_to = 'is_genre') %>%
  filter(is_genre == 1) %>%
  ggplot(aes(reorder(genre, -rating, fun = median), rating)) +
    geom_boxplot(fill = '#132B43') +
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab('Rating') +
  xlab('Genre')
```



Based on the results of the visualizations, the columns with month, days of the month and hour were removed. As previously mentioned, models using the binary genre columns could not be used due to memory allocation problems and, thus, were also removed. Therefore, the data set used in modeling was comprised of the columns listed below. To make code simpler, 'years_since_release' was renamed to 'years' and 'rating_wday' was renamed to 'day'

```
# Select only the necessary columns and print their names
edx = edx %>%
  select(rating,
         movieId,
         userId,
         genres,
         years_since_release,
         rating_wday) %>%
  rename(years = years_since_release,
         day = rating_wday)
names(edx)
```

```
## [1] "rating"  "movieId" "userId"  "genres"  "years"   "day"
```

Modeling

Modeling was carried out by using the holdout method of cross validation (Schneider 1997) with a training set of 90% of the data and testing with the remaining 10% minus the movies and users that were initially in the test set and not in the training set. The following code was used in data partition.

```r
# Set seed to 2020
set.seed(2020)

# Create a partition of 10% of the data and stores the indexes in the
# 'test_index' vector
test_index = createDataPartition(
  y = edx$rating, times = 1, p = 0.1, list = FALSE) %>% .[,1] %>% as_vector()

# Save 90% of the data in the train_set data frame and the remaining in the
# test_set data frame
train_set = edx[-test_index,]
test_set = edx[test_index,]

# Remove movies and users that do not appear in the training set from the test
# set
test_set = test_set %>%
  semi_join(train_set, by = 'movieId') %>%
  semi_join(train_set, by = 'userId')

# Remove unecessary data from memory
rm(test_index)
```

The Root Mean Square Error (RMSE) was used to assess the models by means of the following function.

```r
# Definition of the Root Mean Square Error functionn
RMSE = function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Several trials were carried out to model the data with the train function of the caret package, but all resulted in memory allocation errors. The code was running in a Windows 10 machine with 8GB of RAM. The function memory.limit() turned the operational system unresponsive. To test if the issue was OS-related, the same trials were repeated on a Linux machine (Ubuntu), also with 8GB of RAM. Since the errors persisted and no solutions could be found, only the models described in sequence could be properly constructed and analyzed.

First model

A first model was created to be used as a baseline for comparison. The first model consists in predicting the rating by the average of ratings. Thus, given a movie i and a user j we have $Y = \mu + \epsilon$, being Y the set of predictions, μ the mean of ratings and ϵ the random errors. This model results in a RMSE of 1.060867.

```r
# Predictions from the first model
y_hat1 = mean(train_set$rating)

# RMSE of the first model
RMSE(test_set$rating, y_hat1)
```

```
## [1] 1.060867
```

Second model

The second model includes the effect of movies. As already known and describe in Irizarry (2019), movies are rated differently because of variability in their qualities. Thus, to the second model is added the movie effect (m_i) with m_i being the average of the difference between the mean of ratings and the mean of the particular

movie i, that is $Y = \mu + m_i + \epsilon$ with $m_i = \frac{1}{N_m} \sum_i^{N_m} Y_{j,i} - \hat{\mu}$. The inclusion of the movie effect reduced the RMSE to 0.9440327.

```
# Create the m_i vector with the movie effects
m_i = train_set %>%
  group_by(movieId) %>%
  summarize(m_i = mean(rating - mean(train_set$rating)))

# Add m_i to the train set to be used in the next steps
train_set = train_set %>%
  left_join(m_i, by = 'movieId')

# Add m_i to the test set to be used in the next steps
# Note: The test set is used only to create a vector of predictions with the
# same size as the test set's rating vector, with the corresponding movie ids.
# The ratings of the test set are not used
test_set = test_set %>%
  left_join(m_i, by = 'movieId')

# Remove unecessary data from memory
rm(m_i)

# Predictions from the second model.
y_hat2 = mean(train_set$rating) + test_set$m_i

RMSE(test_set$rating, y_hat2)
```

```
## [1] 0.9440327
```

Third model

The third model adds the effect of users. As already known, different users rate movies differently. User effect is defined as the average rating of the users minus the mean of ratings and the movie effect. The third model is then defined as $Y = \mu + m_i + u_j + \epsilon$ with $u_j = \frac{1}{N_u} \sum_i^{N_u} Y_{i,j} - \hat{\mu} - m_i$. The inclusion of the user effect reduced the RMSE to 0.8657138.

```
# The user effect is defined as the average of ratings by each user minus the
# mean of ratings and the movie effect
u_j = train_set %>%
  group_by(userId) %>%
  summarize(u_j = mean(rating - mean(train_set$rating) - m_i))

# Add u_j to the train set to be used in the next steps
train_set = train_set %>%
  left_join(u_j, by = 'userId')

# Add u_j to the test set to be used in the next steps
test_set = test_set %>%
  left_join(u_j, by = 'userId')

# Remove unecessary data from memory
rm(u_j)

# Predictions from the third model.
y_hat3 = test_set %>%
```

```
  mutate(y_hat = mean(train_set$rating) + m_i + u_j) %>% .$y_hat

RMSE(test_set$rating, y_hat3)
```

```
## [1] 0.8657138
```

Fourth model

The fourth model added genre effects. Initially, the idea was to use the binary variables to do so, which entailed calculating the average rating of each individual genre. However, the code to do so, shown below (Code 1), required more memory than the machine used for analysis could handle, returning a memory allocation error. Thus, the original column genres was used (Code 2), so the effect considered was the effect of the groups of genres. The fourth model is defined as $Y = \mu + m_i + u_j + g_k + \epsilon$ with an effect g for each group of genre, k. The inclusion of genres effect reduced the RMSE to 0.8654029.

Code 1

```
# Code created to calculate the effect of each genre. This code returns "cannot
# allocate vector of size ..." error.
g_i = train_set %>%
  select(rating, m_i, u_i, starts_with('is_')) %>%
  pivot_longer(-c(rating, m_i, u_i), names_to = 'genre', values_to = 'is_genre') %>%
  filter(is_genre == 1) %>%
  group_by(genre) %>%
  summarize(g_i = mean(rating - mean(train_set$rating) - m_i - u_i))
```

Code 2

```
g_k = train_set %>%
  group_by(genres) %>%
  summarize(g_k = mean(rating - mean(train_set$rating) - m_i - u_j))

# Add g_k to the train set to be used in the next steps
train_set = train_set %>%
  left_join(g_k, by = 'genres')

# Add u_j to the test set to be used in the next steps
test_set = test_set %>%
  left_join(g_k, by = 'genres')

# Remove unecessary data from memory
rm(g_k)

# Fourth model: average rating + movie effect + user effect + genres effect
y_hat4 = test_set %>%
  mutate(y_hat = mean(test_set$rating) + m_i + u_j + g_k) %>% .$y_hat

RMSE(test_set$rating, y_hat4)
```

```
## [1] 0.8654029
```

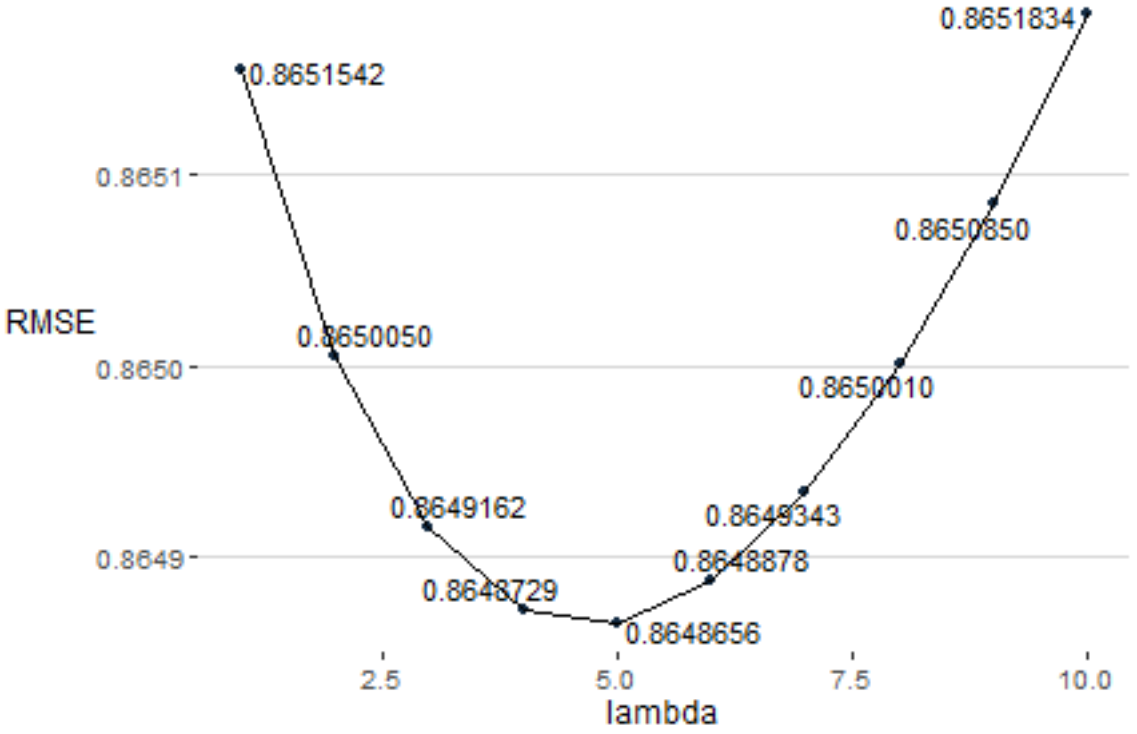
Fifth model

An important characteristic missing in the previous models is regularization. Because the number of ratings varies according to the time the movie is available for release, and because a smaller number ratings results in less precise scores, ratings need to be regularized according to how many ratings they receive. The column

'years' (formerly years_since_release) would be used for this purpose, but using the actual number of ratings is more appropriate, since it accounts for variability of number of ratings independently of its year of release - not all old movies are famous ones, and there is still variability even account for the year of release. The fifth model is the same as the previous one, but with the effects being regularized. So, each effect was calculated by dividing the sums by $N + \lambda$. The best value of λ was selected based on the value that resulted in the smallest RMSE in interval of 1 to 10. This value was $\lambda = 5$, as show in the following plot. With regularization, the RMSE was reduced to 0.8648656.

```
# Iterate from 1 to 10 do select the best lambda value
# For each value of lambda, calculate the regularized predictors, the resulting
# RMSE and stores in the tibble 'lambda_RMSE'
# Note: in the machine used in this analysis, this action took more than three
# hours
lambda_RMSE = sapply(1:10, function(l) {
  # Set lambda to current value of l
  lambda = l

  # Calculate regularized movie effect
  m_i = train_set %>%
    group_by(movieId) %>%
    summarize(m_i = sum(rating - mean(train_set$rating))/(n() + lambda))

  # Update training set with movie effect
  train_set = train_set %>%
    select(-m_i) %>%
    left_join(m_i, by = 'movieId')

  # Update test set with movie effect
  test_set = test_set %>%
    select(-m_i) %>%
    left_join(m_i, by = 'movieId')

  # Remove unecessary data from memory
  rm(m_i)

  # Calculate regularized user effect
  u_j = train_set %>%
    group_by(userId) %>%
    summarize(u_j = sum(rating - mean(train_set$rating) - m_i)/(n() + lambda))

  # Update training set with user effect
  train_set = train_set %>%
    select(-u_j) %>%
    left_join(u_j, by = 'userId')

  # Update test set with user effect
  test_set = test_set %>%
    select(-u_j) %>%
    left_join(u_j, by = 'userId')

  # Remove unecessary data from memory
  rm(u_j)

  # Calculate regularized genres effect
```

```
  g_k = train_set %>%
    group_by(genres) %>%
    summarize(g_k = sum(rating - mean(train_set$rating) - m_i - u_j)/
                (n() + lambda))

  # Update trainning set with genres effect
  train_set = train_set %>%
    select(-g_k) %>%
    left_join(g_k, by = 'genres')

  # Update test set with genres effect
  test_set = test_set %>%
    select(-g_k) %>%
    left_join(g_k, by = 'genres')

  # Remove unecessary data from memory
  rm(g_k)

  # Calculate predictions with regularized predictors
  y_hat5 = test_set %>%
    mutate(y_hat = mean(train_set$rating) + m_i + u_j + g_k) %>% .$y_hat

  # Returns tibble with current lambda value and resulting RMSE
  tibble(lambda = lambda, RMSE = RMSE(test_set$rating, y_hat5))
})

# Plot RMSE against lambda values
lambda_RMSE %>%
  t() %>%
  as_tibble() %>%
  mutate(lambda = unlist(lambda), RMSE = unlist(RMSE)) %>%
  ggplot(aes(lambda, RMSE)) +
  geom_point(color = '#132B43') +
  geom_line() +
  geom_text_repel(aes(label = sprintf("%0.7f", RMSE)))
```



Sixth model

To the sixth and final model the effect of rating week day was added, already regularized with $\lambda = 5$. This predictor was included last because it was not clear if it would actually be a good predictor and reduce the RMSE. The RMSE of the sixth model was 0.865402 with the test set, a increase of 0.0005364 in comparison with the previous model. Consequently, the predictor was not considered good and the previous model, the fifth one, was used in validation, which is discussed in the next section.

```
# Calculate day effect
d = train_set %>%
  group_by(day) %>%
  summarize(d = sum(rating - mean(train_set$rating) - m_i - u_j - g_k)/
              (n() + 5))

# Update the training set with day effect
train_set = train_set %>%
  left_join(d, by = 'day')

# Update the test set with day effect
test_set = test_set %>%
  left_join(d, by = 'day')

# Remove unecessary data from memory
rm(d)

# Predictions of the final model
y_hat6 = test_set %>%
  mutate(y_hat = mean(train_set$rating) + m_i + u_j + g_k + d) %>% .$y_hat
```

```r
# RMSE of final model
RMSE(test_set$rating, y_hat6)
```

```
## [1] 0.865402
```

The following code is the same used in the function to select the best value of λ, but with the value set to 5. This step is necessary to generate the predictions of the fifth model that will be analyzed in the next section.

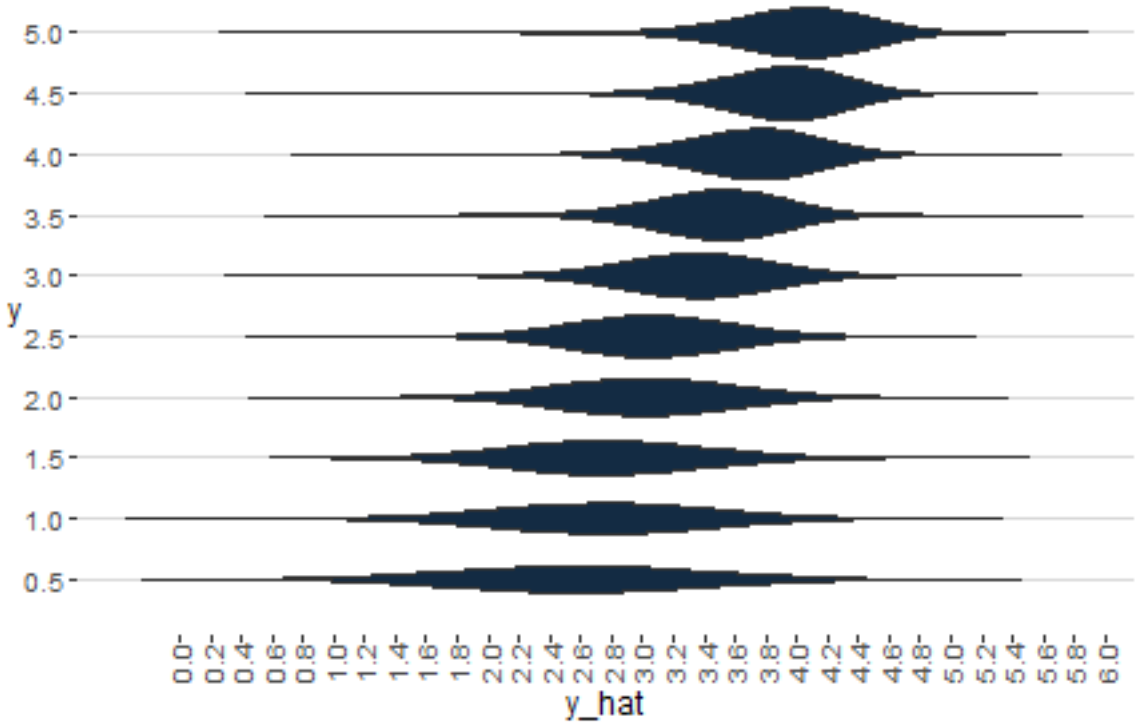
```r
# Calculate regularized movie effect
m_i = train_set %>%
  group_by(movieId) %>%
  summarize(m_i = sum(rating - mean(train_set$rating))/(n() + 5))

# Update training set with movie effect
train_set = train_set %>%
  select(-m_i) %>%
  left_join(m_i, by = 'movieId')

# Update test set with movie effect
test_set = test_set %>%
  select(-m_i) %>%
  left_join(m_i, by = 'movieId')

# Remove unecessary data from memory
rm(m_i)

# Calculate regularized user effect
u_j = train_set %>%
  group_by(userId) %>%
  summarize(u_j = sum(rating - mean(train_set$rating) - m_i)/(n() + 5))

# Update training set with user effect
train_set = train_set %>%
  select(-u_j) %>%
  left_join(u_j, by = 'userId')

# Update test set with user effect
test_set = test_set %>%
  select(-u_j) %>%
  left_join(u_j, by = 'userId')

# Remove unecessary data from memory
rm(u_j)

# Calculate regularized genres effect
g_k = train_set %>%
  group_by(genres) %>%
  summarize(g_k = sum(rating - mean(train_set$rating) - m_i - u_j)/
              (n() + 5))

# Update trainning set with genres effect
train_set = train_set %>%
  select(-g_k) %>%
  left_join(g_k, by = 'genres')
```

```r
# Update test set with genres effect
test_set = test_set %>%
  select(-g_k) %>%
  left_join(g_k, by = 'genres')

# Remove unecessary data from memory
rm(g_k)

# Calculate predictions of final model
y_hat5 = test_set %>%
  mutate(y_hat = mean(train_set$rating) + m_i + u_j + g_k) %>% .$y_hat
```

Results

With the fifth model, RSME decreased to a value only 0.0000344 smaller than 0.86490, using the test set. Before reporting the RMSE with the validation set, some considerations are worth to be drawn regarding the predictions. The following violin plot shows the distribution of the predictions for each rating. Two clear patterns can be observed. The first one is a tendency of the algorithm to underestimate higher values and overestimate lower ones. The second is the higher precision for higher rates and lower precision for lower rates.

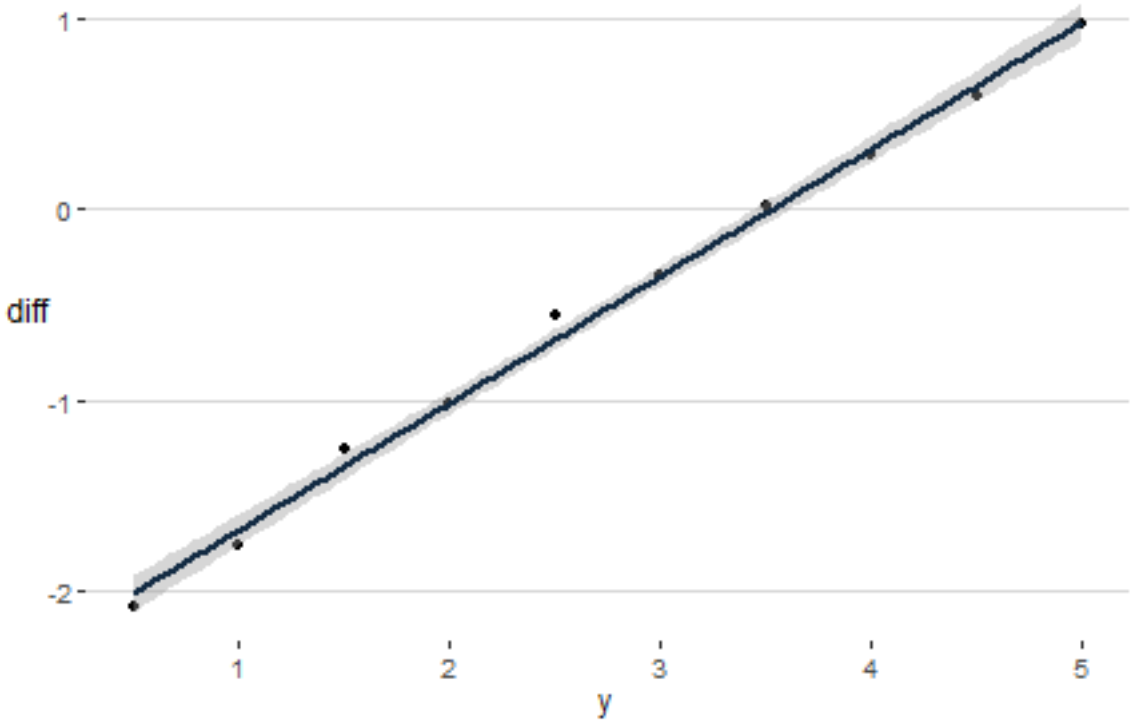
```r
# Generate violin plot of ratings of the test set against predictions from the
# fifth model
tibble(y_hat = y_hat5, y = test_set$rating) %>%
  ggplot(aes(y_hat, y, group = y)) +
  geom_violin(fill = '#132B43') +
  scale_x_continuous(breaks = seq(0, 6, .2)) +
  scale_y_continuous(breaks = seq(0, 5, .5)) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



The difference between predicted values and the real values have a strong linear relationship with the rating, so probably there are some effect not accounted for by the current algorithm that should be included in the model.

```
tibble(y_hat = y_hat5, y = test_set$rating) %>%
  group_by(y) %>%
  summarize(median = median(y_hat)) %>%
  mutate(diff = y - median) %>%
  ggplot(aes(y, diff)) +
  geom_point() +
  geom_smooth(method = 'lm', color = '#132B43')
```

```
## `geom_smooth()` using formula 'y ~ x'
```



To validate the algorithm, the effects were calculated again using the whole edx data set and not only the training set used in modeling. The optimized λ was used (5) and the final RMSE is 0.8644501, 0.0004499 smaller than 0.86490.

```
# Calculate movie effect with the whole data set
m_i = edx %>%
  group_by(movieId) %>%
  summarize(m_i = sum(rating - mean(edx$rating))/(n() + 5))

# Update data set with movie effect
edx = edx %>%
  left_join(m_i, by = "movieId")

# Update validation set with movie effect
validation = validation %>%
  left_join(m_i, by = "movieId")

# Remove unecessary data from memory
rm(m_i)

# Calculate user effect with the whole data set
u_j = edx %>%
  group_by(userId) %>%
  summarize(u_j = sum(rating - mean(edx$rating) - m_i)/(n() + 5))

# Update data set with user effect
edx = edx %>%
  left_join(u_j, by = "userId")

# Update validation set with user effect
```

```
validation = validation %>%
  left_join(u_j, by = "userId")

# Remove unecessary data from memory
rm(u_j)

# Calculate genres effect with the whole data set
g_k = edx %>%
  group_by(genres) %>%
  summarize(g_k = sum(rating - mean(edx$rating) - m_i - u_j)/(n() + 5))

# Update data set with genres effect
edx = edx %>%
  left_join(g_k, by = "genres")

# Update validation set with genres effect
validation = validation %>%
  left_join(g_k, by = "genres")

# Remove unecessary data from memory
rm(g_k)

# Calculate final predictions
y_hat_final = validation %>%
  mutate(y_hat = mean(edx$rating) + m_i + u_j + g_k) %>% .$y_hat

RMSE(validation$rating, y_hat_final)
```

```
## [1] 0.8644501
```

Conclusion

This report summarized the development of an algorithm that predicts movie ratings from a subset of the movie lens data set considering variability by user, movie, genre (genre combination) and day of the week of rating. The final RMSE with the validation set was [...]. Although the algorithm have reached a desired RMSE, some issues were found during development and there are still room for improvement. These questions are summarized in the next section.

Limitations and suggestions for future work

The main limitation of the present work was the impossibility of using more computationally demanding methods due to memory allocation problems. A search for the error message "cannot allocate vector of size" in stackoverflow.com results 275 r-related questions, which shows that the problem is somehow common. All the efforts to solve the problems were unfruitful, but future work may bring solutions to this issue. Concerning the final model, as was already mentioned, the predictions overestimate small values and underestimate larger ones. Identifying possible causes for this pattern can lead to better predictions. Finally, predictions are more precise with higher ratings. Investigating this pattern can also lead to a better fine tuned model and, thus, better predictions.

References

Arnold, Jeffrey B. 2019. *Ggthemes: Extra Themes, Scales and Geoms for 'Ggplot2'*. https://CRAN.R-project.org/package=ggthemes.

Bache, Stefan Milton, and Hadley Wickham. 2014. *Magrittr: A Forward-Pipe Operator for R.* https://CRAN.R-project.org/package=magrittr.

Bååth, Rasmus. 2018. *Beepr: Easily Play Notification Sounds on Any Platform.* https://CRAN.R-project.org/package=beepr.

Grolemund, Garrett, and Hadley Wickham. 2011. "Dates and Times Made Easy with lubridate." *Journal of Statistical Software* 40 (3): 1–25. http://www.jstatsoft.org/v40/i03/.

Irizarry, Rafael A. 2019. *Introduction to Data Science: Data Analysis and Prediction Algorithms with R.* Leanpub. https://leanpub.com/datasciencebook.

Kuhn, Max. 2020. *Caret: Classification and Regression Training.* https://CRAN.R-project.org/package=caret.

R Core Team. 2018. *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org.

Schneider, Jeff. 1997. "Cross Validation." *Carnegie Mellon School of Computer Science.* Carnegie Mellon University. https://www.cs.cmu.edu/~schneide/tut5/node42.html.

Slowikowski, Kamil. 2020. *Ggrepel: Automatically Position Non-Overlapping Text Labels with 'Ggplot2'.* https://CRAN.R-project.org/package=ggrepel.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.