Movielens Recommendation Project

Juan Eloy Suarez - PH125 Data Science - Harvard

16/May/2021

1 Introduction

Executive Summary

In this project, we want to create a movie recommendation system. More specifically, we need to predict the value an user will rate a movie in a test set we will use for validation, based on a given set of users and movie ratings.

Our data is from the MovieLens 10M dataset (10 million ratings). According to the definition of the challenge, we have a training dataset `edx` of approximately 9 million records, and a test dataset `validation` of 1 million records. The true rating column in the validation set will be used to compare the predictions of the model, which will me measured according to its `RMSE` (Root Min Square Error).

The prediction algorithm used in this project departs from the basic approach used in the course (Bellkor's approach), consisting in applying to the generic average prediction an user and a movie *bias* (statistical effect) for each observation to predict, which is regularized for prevalence compensation.

However, our modelling adds some originality to enrich this approach:

- Calculates, includes and **regularizes biases for Genre/s and Year** of the movie
- Extracts **Sentiments induced from Title** of each movie and includes and regularizes also their bias
- **Ensembles a second model** that, using first model predictions as input, improves prediction by including biases related with the **moment when rating is done**, i.e. **Weekday**, **Month** and year between rating and movie release (**Age-At-Rating**). *Note this second model is sequential with first one, so it could be discarded if no rating moment available for prediction, and reaching anyway a good enough RMSE.*

We payed special attention to **ensure that test data are never used for training the model, or even estimating parameters** as λ for regularization. For this reason, we create a `sub-partitions` for parameter training and test, extracted exclusively from the training set (`edx`).

With all of this, an already acceptable RMSE of **0.8643** is achieved already by first model, and improved by second model up to **0.8639**.

2 Data Exploration

Preparation of environment, train&test partition

First of all, we need to initialize environment, install required packages and download dataframes `ratings` and `movies`:

```
names(ratings)
```

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

```
names(movies)
```

```
## [1] "movieId" "title"   "genres"
```

In preparation for later use, we extract as new prescriptor the Release Year information contained in brackets in the title of each movie:

```
# Separating Year form title to get a new prescriptor
movies <- movies %>%
  mutate (year=as.numeric(str_match(title, "(^.*)\\s[(](\\d{4})[)]$")[,3]),
          title=str_match(title, "(^.*)\\s[(](\\d{4})[)]$")[,2])
```

Once prepared data we need, we create the train (`edx`) and test (`validation`) separated partitions based on `movielens` data:

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

This analysis will make use of the root mean-square error (RMSE). We need to create a function to be called and calculate RMSE for each one of the models we are going to test. We will also calculate the global average of rating, which will be useful as starting point of our analysis and modelling:

```
# Function to calculate RMSE
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
# Average of all ratings
mu <- mean(edx$rating)
```

Understanding the data

Exploration of the dataset shows a dataframe that contains all ratings (numbers between 0.5 and 5 in intervals of 0.5) made by users to movies. Dataset includes also information about the movie and the transaction timestamp, that we'll convert in date using `lubridate` package. Year is originally included in the title, so we extracted it into a different feature.

```
str(edx)
```

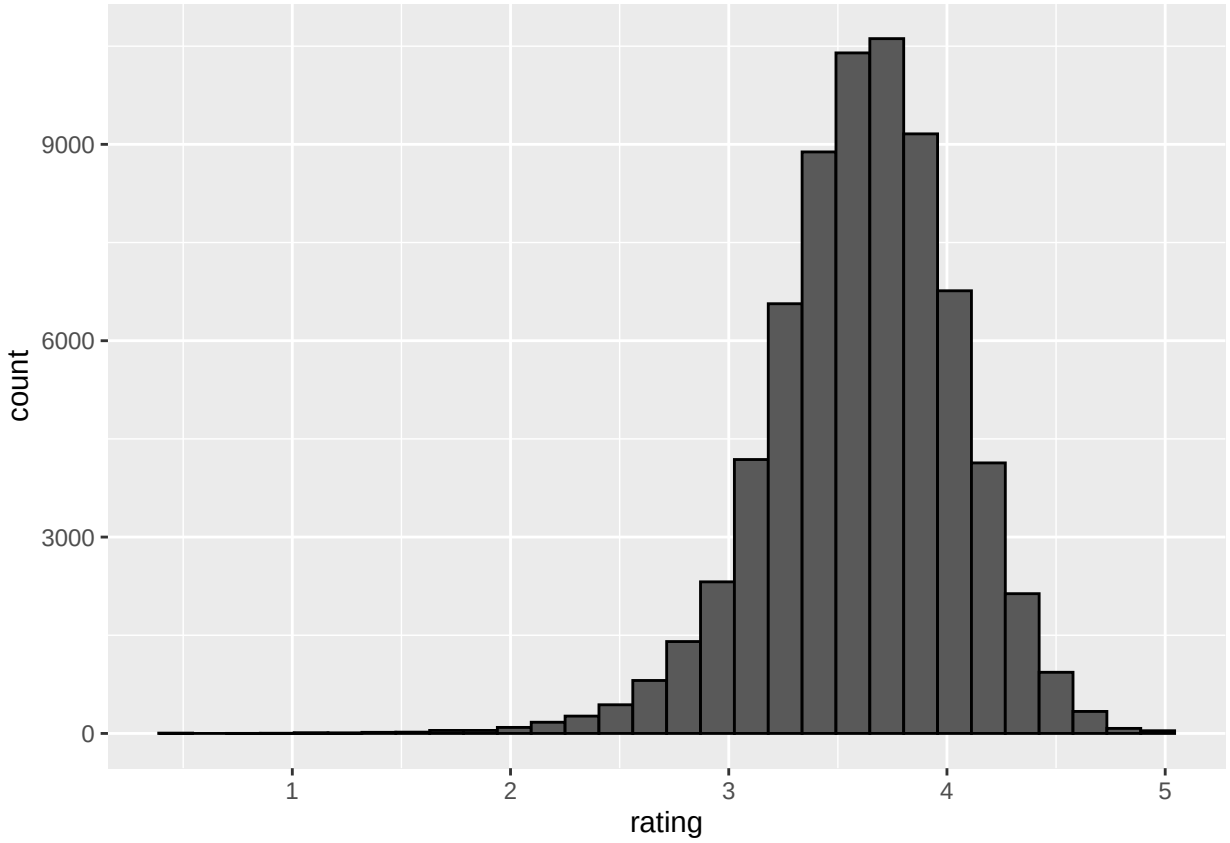
```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  7 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang" "Net, The" "Outbreak" "Stargate" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
##  $ year     : num  1992 1995 1995 1994 1994 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

There are three sources of information for our exploratory analysis:

- User-related information (ratings by users)
- Movie-related information (ratings received, title, genre, release year)
- Rating-related information (when exactly rating transaction happens)

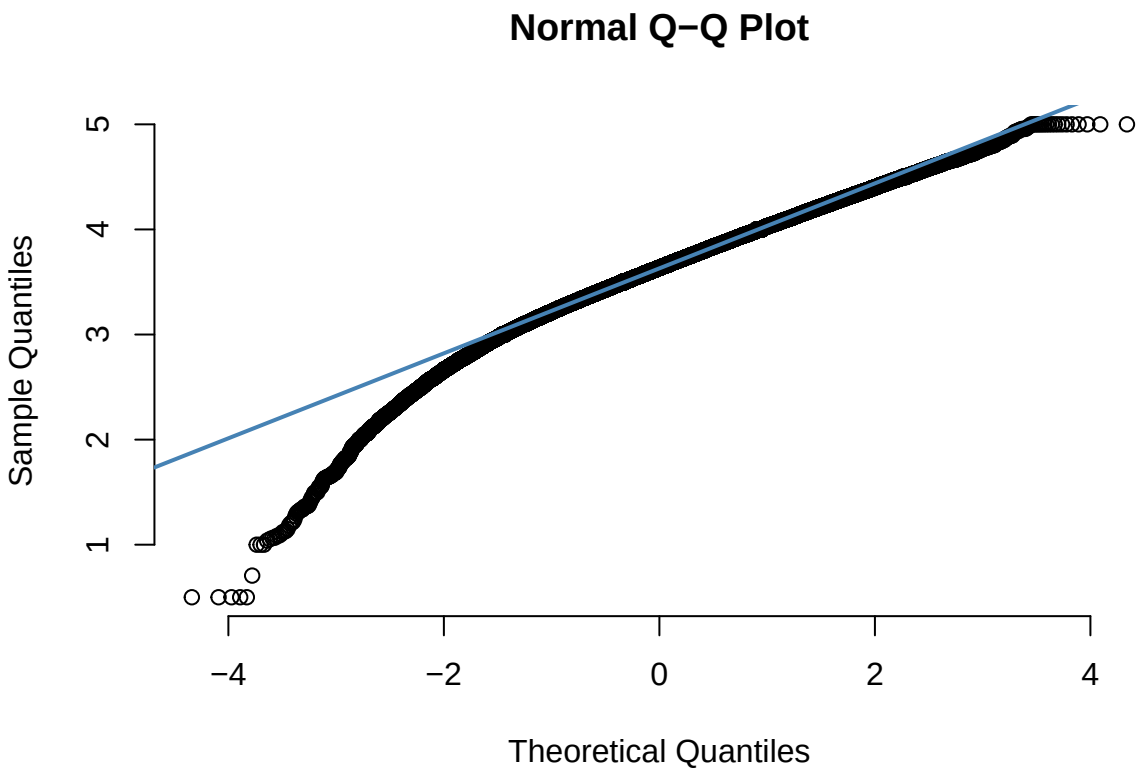
User related data

Total number of users is 69878 and movies is 10677

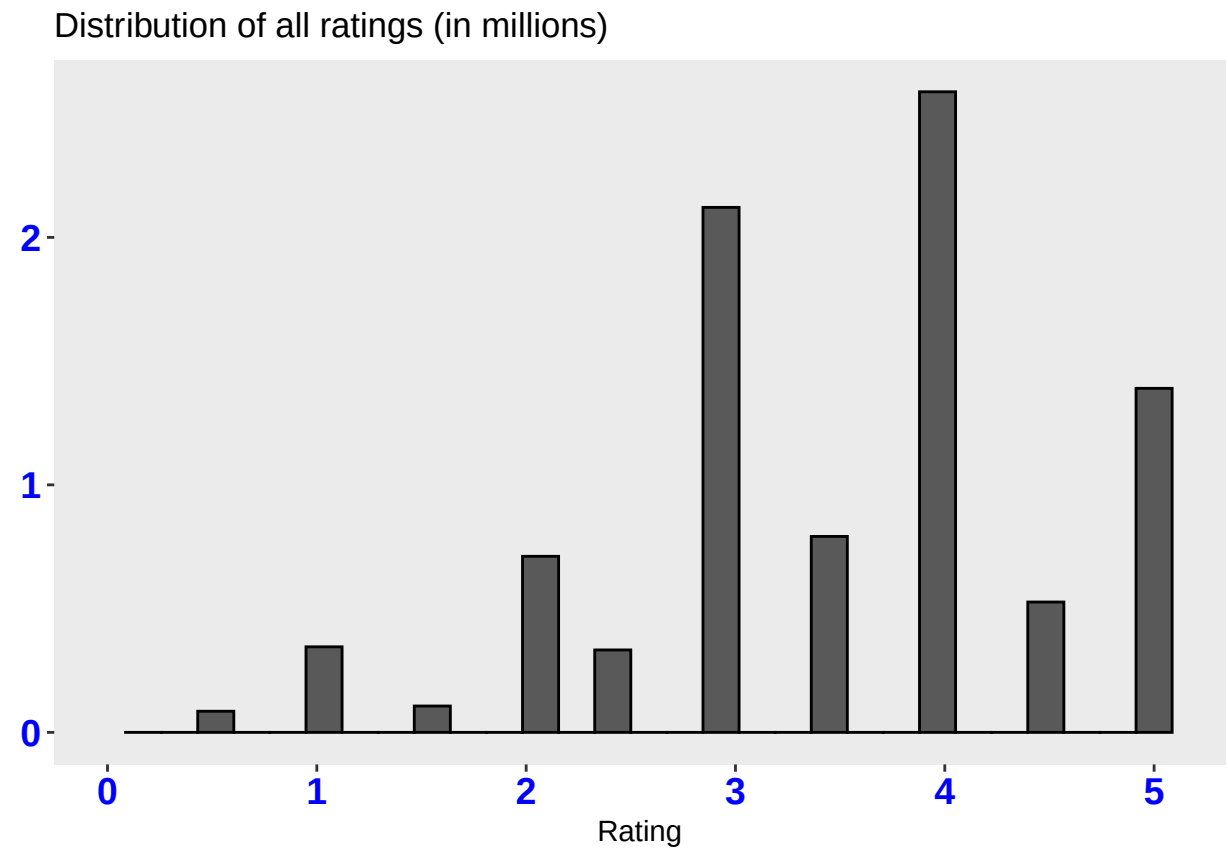


Distribution of user's ratings is around an average of **mu**=3.51247, with a standard deviation of 1.06033, and we see an approximate normal distribution of the ratings by user in the QQ-Plot:

```
# Normality test by user
qqnorm(byUser_ratings$rating, pch = 1, frame = FALSE)
qqline(byUser_ratings$rating, col = "steelblue", lwd = 2)
```



Distribution of all ratings per rating category shows us this pattern:



Movie related data

Dataset contains interesting information on how rating depends on the movie. Looking in detail at the prescriptors we have related with the movie, we can obtain some interesting insights. The MOVIE bias consists in a generic "quality" effect that we can assign to each movie. It can be estimated by the mean of all ratings received by that movie, and we can use it to refine the generic prediction.

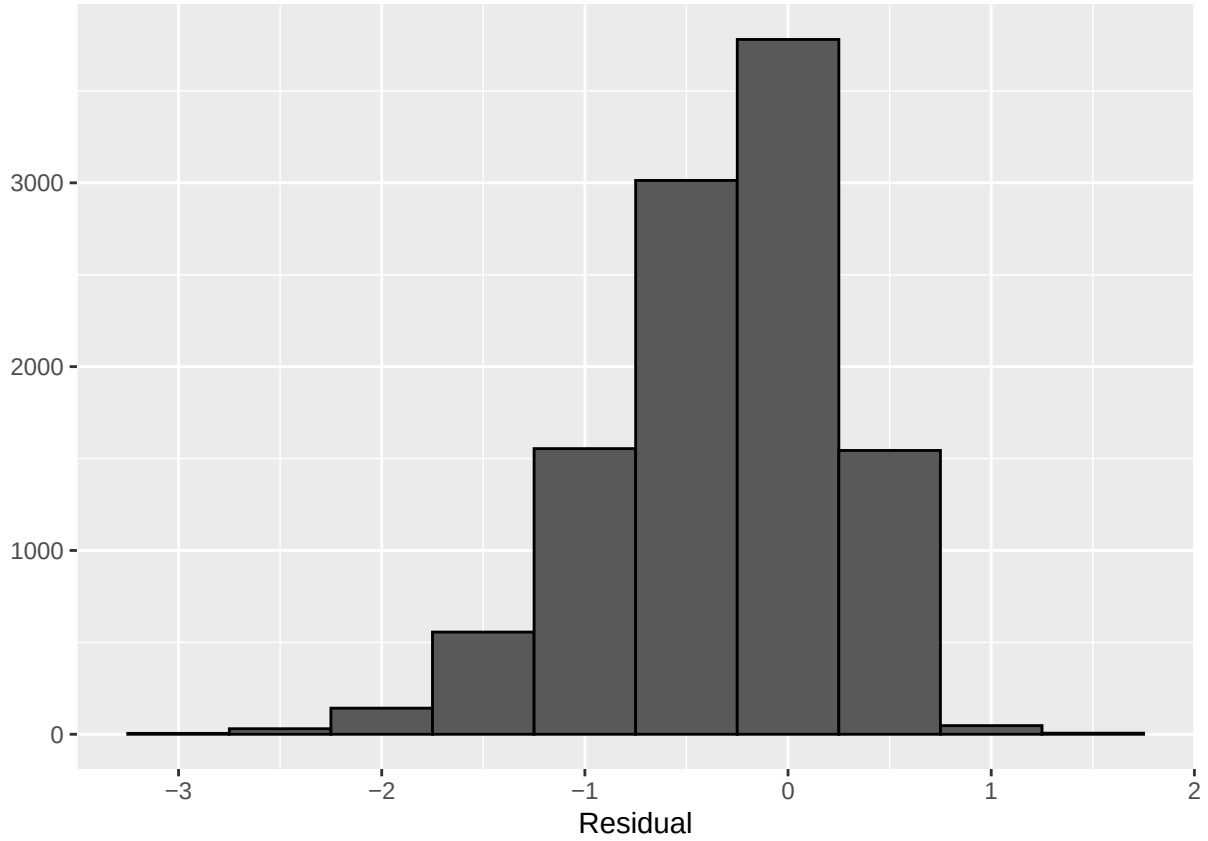
```
byMovie_ratings <- edx %>%
  group_by(movieId) %>%
  summarize(rating=mean(rating), Residual = mean(rating - mu))
head(byMovie_ratings)
```

```
## # A tibble: 6 x 3
##   movieId rating Residual
##     <dbl>  <dbl>    <dbl>
## 1       1   3.93    0.415
## 2       2   3.21   -0.307
## 3       3   3.15   -0.365
## 4       4   2.86   -0.648
## 5       5   3.07   -0.444
## 6       6   3.82    0.303
```

Now, we have each movie assigned to a single rating that is calculated as the average of all ratings to that movie:

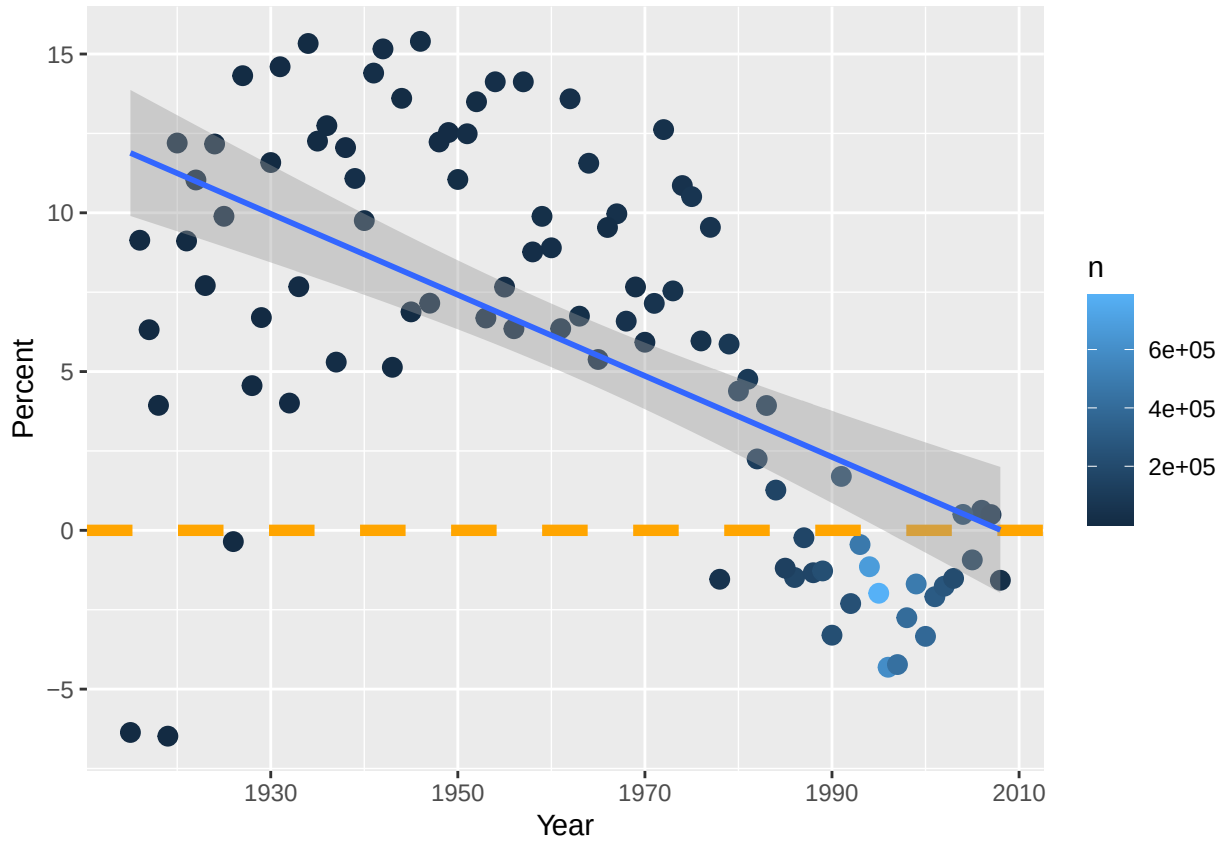
```
mu_byMovie <- mean(byMovie_ratings$rating)  # Average of ratings movie to movie
```

We see now that the average of these movie averages 3.19174 is significantly less that the direct average of all ratings μ. Interpretation of this is that there is a significant difference in the number of ratings each movie receives (prevalence), and movies with many ratings tend to have higher ratings. This effect can be also deducted from the analysis of residual ratings (vs. μ) of the movie's ratings, where we see an unbalanced-to-left distribution:



Above variability on prevalence recommends we use regularization to compensate too big differences of number of ratings per movie.

The YEAR a movie was released seems to have some influence in the rating a movie gets, so we can consider this year-effect as an added bias to apply (discount) when building our model.

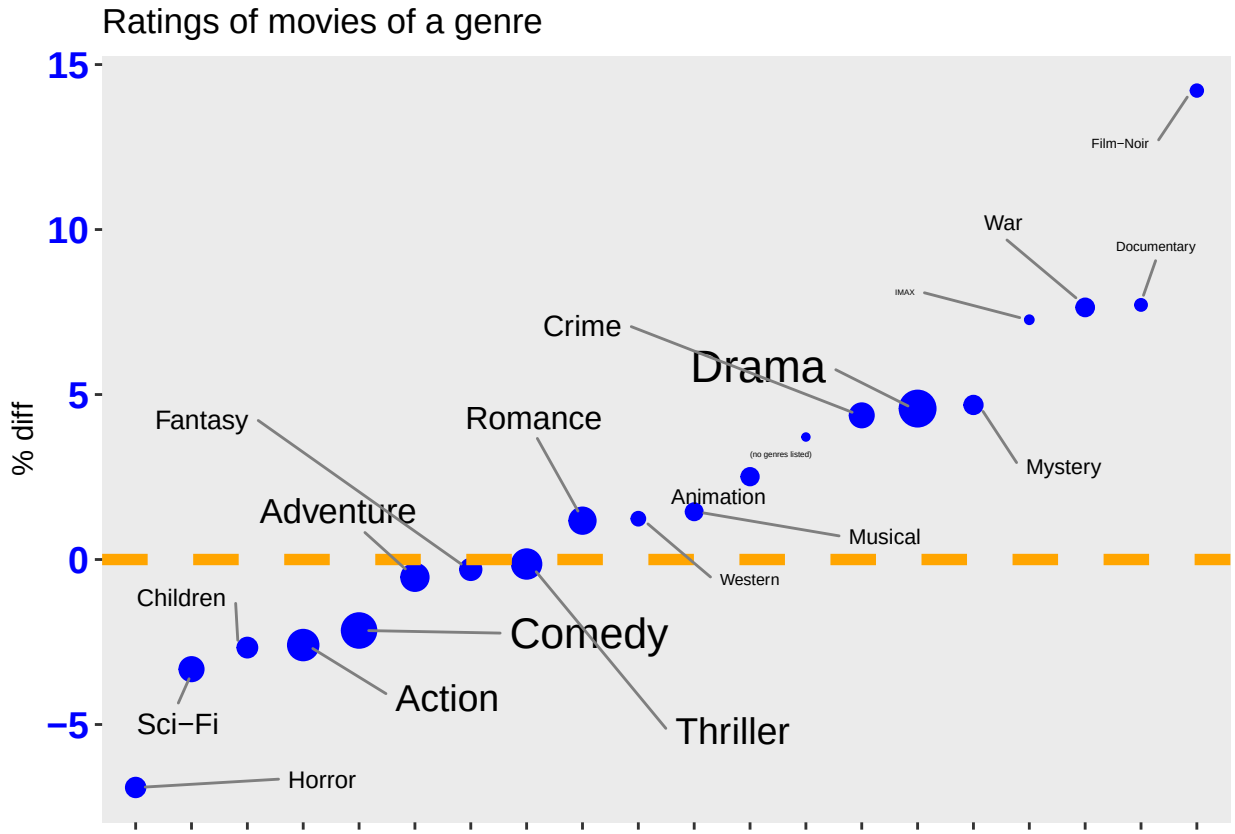


We see outliers in some years with few ratings, but also a trend overtime to rate lower to more recent movies, even considering these years have many more ratings. Again, a regularization is recommended to compensate this effect and isolate, as much as possible the bias we clearly see associated to the year a movie was released.

Our dataset includes, associated to each movie, the list of GENRES it is assigned to. In order to be more precise we'll use this code to extract specific gender/s by movie, instead of the list.

```
# GENRE (individualized) bias
tmp <- setNames(strsplit(as.character(edx$genres), split = "[|]"), edx$rating)
indiv_genres <- data.frame(genre = unlist(tmp), rating = as.numeric(as.character(rep(names(tmp), sapply
byIndivGenre_ratings <- indiv_genres %>%
  group_by(genre) %>%
  summarize(b_g = mean(rating) - mu, n=n(), Percent=100*b_g/mu) %>%
  mutate(genre=reorder(genre,b_g,FUN=median)) %>%
  arrange(desc(abs(Percent)))
rm(tmp, indiv_genres)
# byIndivGenre_ratings
```

This clearly shows a genre-bias exist:



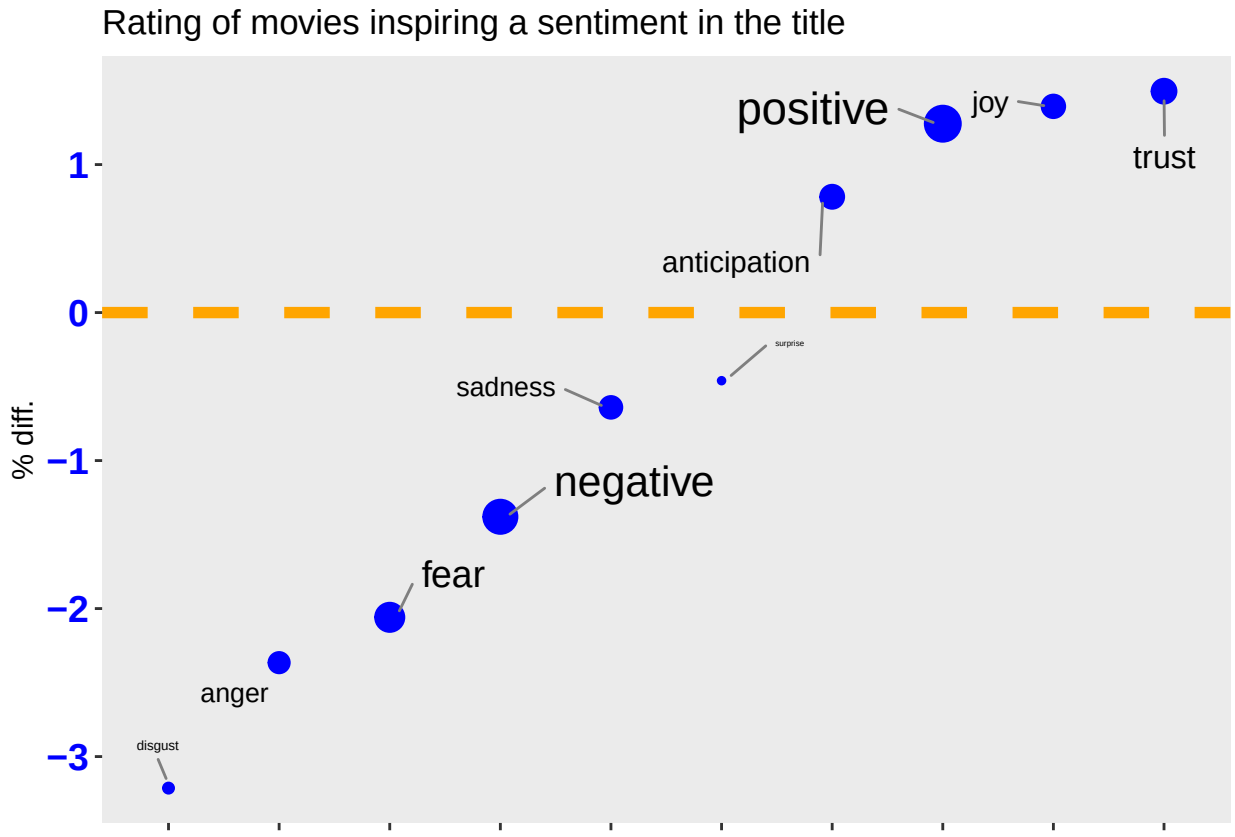
Variability in prevalence also recommends us here to regularize when applying this bias.

Now, we will go one step further and use the TITLE of the movie as prescriptor. It seems clear that just the title is not a significant or useful source of bias or causation; however, we will convert this wording in SENTIMENTS. This means that some words can be associated to sentiments or emotions, that we can associate a movie. Then, maybe this title-sentiment characteristic of the movie could be a bias we apply. I.e. are movies containing words like "dead", "murder" higher rated that others containing "happiness" o "luck"? For exploring this, we will use lexicons to convert words in title into sentiments and link it to ratings. This code performs it based on public `lexicons`:

```
# Sentiments based on TITLE bias
# Extracting words contained in each movie's title
pattern <- "([^A-Za-z\\d#@']|'(?![A-Za-z\\d#@]))"
movie_words <- edx %>%
  group_by(movieId) %>%
  summarize(title = first(title), rating=mean(rating)) %>%
  mutate(title = str_replace_all(title, "[(+)]", "")) %>%
  unnest_tokens(word, title, token = "regex", pattern = pattern) %>%
  select(movieId, word, rating)
# Specific lexicons to try
# Identifies EMOTIONS (NRC) : {anger, anticipation, disgust, fear, joy, negative, positive, sadness, su
myLexicon_nrc <- get_sentiments("nrc") %>% select(word, sentiment)
# Identifies LEVELS (BING): {positive, negative}:
myLexicon_bing <- get_sentiments("bing") %>% select(word, sentiment)
# Calculate ratings of moving containing a specific sentiment based on NRC Lexicon
bySentiment_ratings_nrc <- movie_words %>%
  inner_join(myLexicon_nrc, by = "word") %>%
```

```
  group_by(sentiment) %>%
  summarize(b_sr = mean(rating)-mu_byMovie, n=n()) %>%
  mutate(sentiment=reorder(sentiment,b_sr,FUN=median), Percent=100*b_sr/mu_byMovie)
# Calculate ratings of moving containing a specific sentiment based on BING Lexicon
bySentiment_ratings_bing <- movie_words %>%
  inner_join(myLexicon_bing, by = "word") %>%
  group_by(sentiment) %>%
  summarize(Residual = mean(rating)-mu_byMovie, n=n()) %>%
  mutate(sentiment=reorder(sentiment,Residual,FUN=median), Percent=100*Residual/mu_byMovie)
```

We visualize differences:



When applying just positive/negative sentiment we get also a bias:

```
## # A tibble: 2 x 4
##   sentiment Residual     n Percent
##   <fct>        <dbl> <int>   <dbl>
## 1 negative   -0.0655  1908  -2.05
## 2 positive    0.0141  1242   0.441
```

We will now calculate impact of all sentiments detected per movie. For simplification, and looking at its significance, we use positive/negative lexicon, by summarizing all +1 for positives or -1 for negatives.

```
# To apply to each movie, we summarize per movie positive or negative sentiments detected
byMovie_SentimentEffect <-
```

```
  (movie_words %>% inner_join(myLexicon_bing, by = "word")) %>%
  left_join(bySentiment_ratings_bing, id = "sentiment") %>%
  mutate(weight=if_else(sentiment=="negative", -1, 1)) %>%
  group_by(movieId) %>%
  summarize(weightSentiment = sum(weight))
head(byMovie_SentimentEffect)
```

```
## # A tibble: 6 x 2
##   movieId weightSentiment
##     <dbl>           <dbl>
## 1       3              -1
## 2       9              -1
## 3      12               0
## 4      15              -1
## 5      29              -1
## 6      31              -1
```

We observe differences for some specific sentiments, when present, and in general a behavior where negative detected sentiments penalize the rating of a movie. Our summarization per movie results in a range from very negative (many more negatives than positives in title), to very positive.

```
# Range from very negative to very positive sentiments per movie
range(byMovie_SentimentEffect$weightSentiment)
```

```
## [1] -4  3
```

This will be our sentiment-bias by movie and we will and also regularize it, since prevalence varies significantly among sentiments, but also compared to those movies with no sentiment assigned, which are majority.

Rating-related information

Finally, we are going to explore some other variables we can extract from the timestamp information collected in each rating. We can easily convert it to date using `lubridate` package, and the determine if there are biases related with when exactly a rating is registered... Are ratings entered on Tuesday lower or higher that of weekends? Have winter months higher or lower ratings than summer ones? Does the "age" of a movie (how long ago was released when entering the rating) affect to the rating it gets? (or, for example, are just released movies rated higher/lower that older ones?)

Let's focus initially in the DAY OF THE WEEK by averaging ratings:

```
# ================
# WEEKDAY bias
byWeekday_ratings <- edx %>%
  mutate(WeekDay = factor(weekdays(as_datetime(edx$timestamp)))) %>%
  group_by(WeekDay) %>%
  summarize(b_wk = mean(rating) - mu, n=n(), Percent=round(100*b_wk/mu, digits = 2)) %>%
  arrange(Percent) %>%
  select(WeekDay, n, Percent)
byWeekday_ratings
```

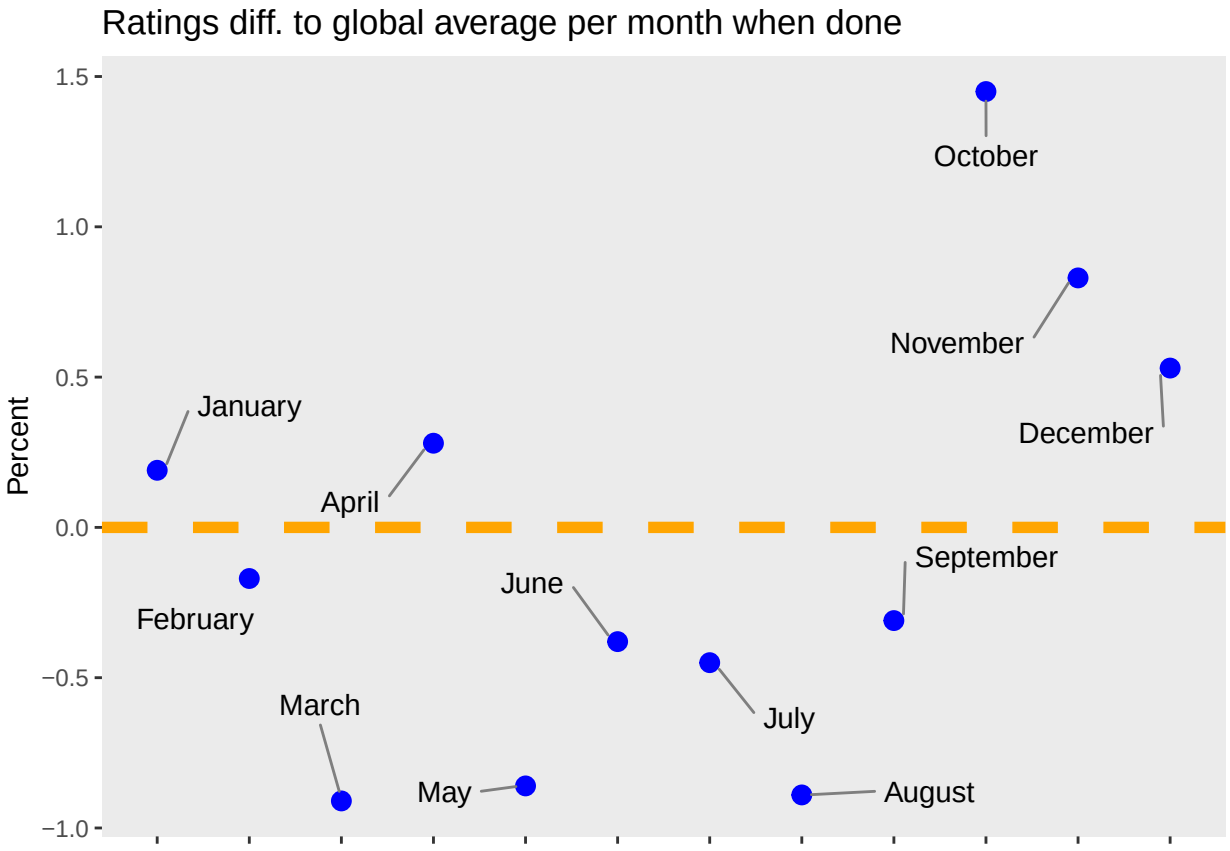
```
## # A tibble: 7 x 3
```

```
##   WeekDay         n Percent
##   <fct>       <int>   <dbl>
## 1 Thursday  1241493   -0.33
## 2 Wednesday 1332505   -0.31
## 3 Tuesday   1431735   -0.05
## 4 Friday    1232250    0.01
## 5 Monday    1410155    0.13
## 6 Sunday    1226472    0.16
## 7 Saturday  1125445    0.47
```

Although this effect is not very significant (always less than 0.5%), we see some bias, and with a prevalence very homogeneous, so we can consider this prescriptor for refining our model, despite it has not a big causation.

Following a similar approach, we will dig now in the MONTH when the rating is done:

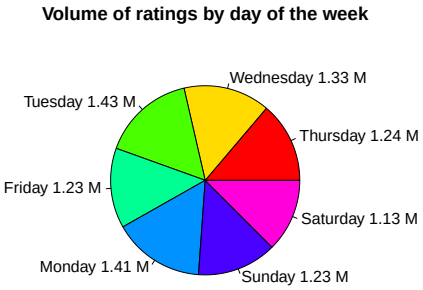
```
# ================
# MONTH bias
byMonth_ratings <- edx %>%
  mutate(Month = month(as_datetime(timestamp), label = TRUE, abbr = FALSE)) %>%
  group_by(Month) %>%
  summarize(b_m = mean(rating) - mu, n=n(), Percent=round(100*b_m/mu, digits = 2)) %>%
  arrange(Percent) %>%
  select(Month, n, Percent)
```

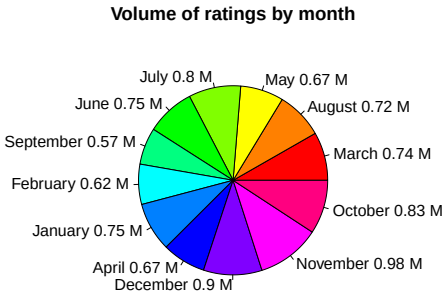


We appreciate now that Month has some significance as bias, since it shows in most cases month-effects

higher than 0.5%, even closer to 1.5% in October. We will also consider this prescriptor for refinement of our modelling.

Displaying prevalences of WeekDay and Month prescriptors we see approximately homogeneous distribution among categories:

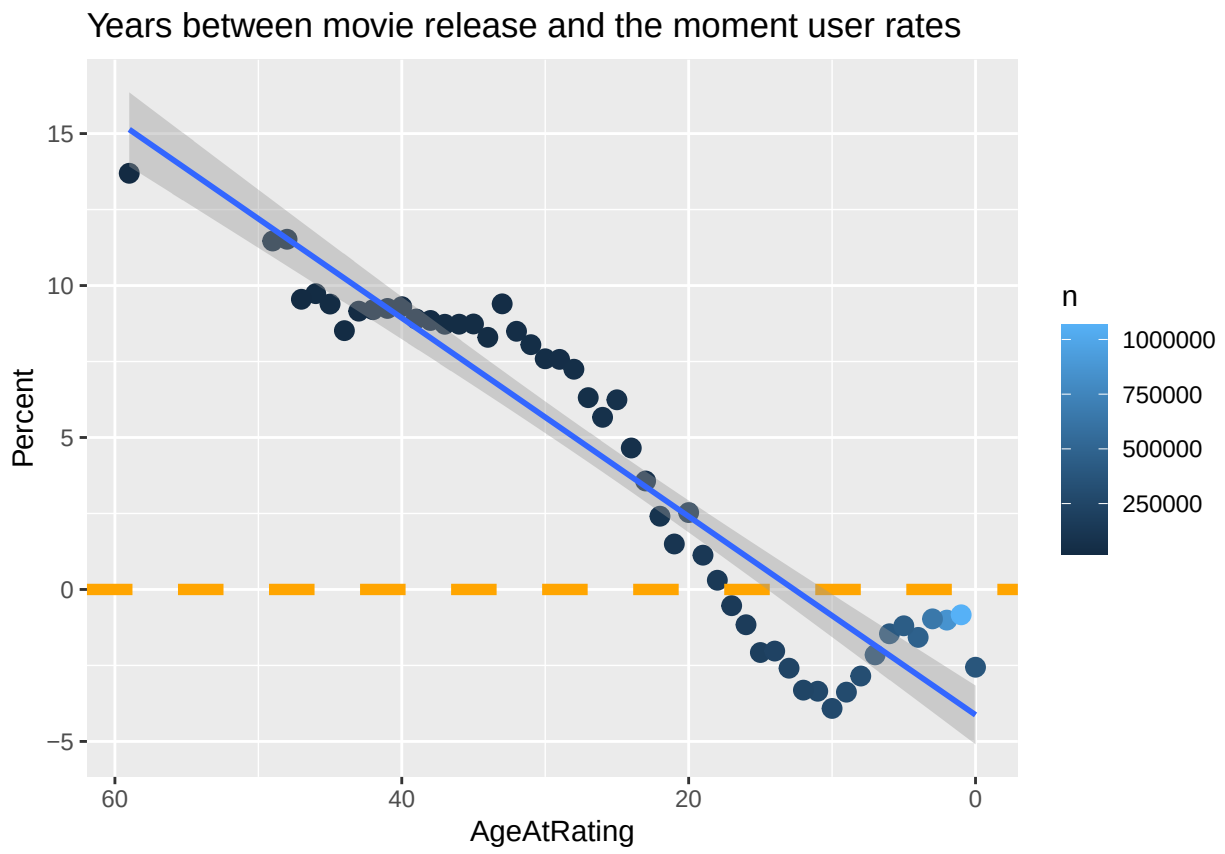




Now we are going to analyze the difference in years between the exact moment when rating is registered and the release year of the movie, we'll call this the AGE-AT-RATING. We previously extracted Year feature from the title of the movie. This code calculates this period of time in years, adding a filter to exclude Ages with not significant numbers of observations, which might behave as exceptional/outliers cases:

```
# Explore AGE bias
minRecords <- nrow(edx) / (5*100) # To avoid non-significant low prevalence cases, let's exclude ages w
byAgeRating_ratings <-  edx %>%
  mutate(YearRating = year(as_datetime(timestamp)), AgeAtRating = YearRating - year) %>%
  group_by(AgeAtRating) %>%
  summarize(b_ag = mean(rating) - mu, n=n(), Percent=100*b_ag/mu) %>%
  filter(!is.na(AgeAtRating) & (AgeAtRating >= 0) & (n > minRecords)) %>%
  arrange(desc(abs(Percent))) %>% select(AgeAtRating, n, Percent)
```

It graphically shows:



Looking at the chart, it seems clear that there is a bias here (more than 13% in some *ages*). Lowest ratings are assigned to movies 10 years old when rating, and in general movies are better rated when longer since rating moment. Despite chart seems similar to the Release Year prescriptor, it is important to distinguish that in this case we are assessing "how old" is the movie when the rating is registered. This can be significantly different than how old the movie is, since in first case depends on when the rating is created.

As stated above, looking at the distributions of Weekday, Month and Age-at-Rating prescriptors, we see they are reasonably homogeneous, except for Age-At-Rating. However, for simplicity, and taking into consideration we are in a refinement phase will very small effect values and that we already filtered out Ages with few support, we will not proceed to regularize bias correction for these last three prescriptors, that will be modelled in a separate, ensembled model onto the Phase 1 Model.

Methods & Analysis

In this section, we will explain the process and techniques used, including data cleaning, data exploration and visualization, insights gained, and your modeling approach.

The prediction algorithm used in this project generally follows the simple model used in the course judging the "bias" or difference from the mean for each user, item, and genre and other prescriptors to finally implement a regularization to discount extreme, occasional values. This is performed in **two phases**.

Phase 1 model (based on user, movie data). The prediction strategy followed departs from the basic estimation of the bias or statistical effect of user and movie. From this point, we add movie genre, movie year and sentiment/emotion extracted from the words title of the movie (positive/negative) to the set of categories to *unbias*, resulting in a regularized model including bias ("difference from the mean") following features: user, movie, genre, year and sentiment.

Phase 2 model (based on Phase1 predictions and rating event data). After obtaining a significantly improved `RMSE` using approach above, we still find an opportunity to reduce it. We will ensemble a second model to train, based on results of Phase 1, using prescriptors extracted from the "rating event", i.e. when the rating is done (day of week, month and time in years between movie's year and the rating event).

Phase 1 model

The model used for developing the prediction algorithm follows this approach: the mean rating μ is modified by one or more *bias* terms b with a residual error ϵ expected.

```
mu <- mean(edx$rating)  # We use mean 'mu' as starting point to advance in effects removal
```

$$Y_{u,i,g,y,s} = \mu + b_i + b_u + b_g + b_y + b_s + \varepsilon_{i,u,g,y,s}$$

According to above, we start with a trivial model based in predicting always de average. This results in a pretty high RMSE that we are going to progressively improve:

Method	RMSE
Mean as base reference	1.0612

Data preparation for improving prediction Since there are some prescriptors not explicitly present in our dataset, we need to do some preprocessing to allow availability of Year and Sentiment features. For the year, we extracted the four digits out of the movie's title. This operation was done during initial phases of data preparation/partition.

```
names(edx)
```

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

In order to assign sentiments for each movie, we use the title, extract words contained and thus we can, for some of them, assign positive/negative scores summarizes each movie's detected emotions, based on BING public lexicon (collection of assignments between words and positive/negative semantic for each). During data exploration we learnt that this title word might be also a useful bias to discount, and created a dataframe to support this calculation.

The are several available lexicons, we are using the BING for simplicity, and based in the fact differences with other (NRC) that assign a wider range of emotions, will be not significant quantitatively.

```
head(bySentiment_ratings_bing)
```

```
## # A tibble: 2 x 4
##   sentiment Residual     n Percent
##   <fct>        <dbl> <int>   <dbl>
## 1 negative   -0.0655  1908  -2.05
## 2 positive    0.0141  1242   0.441
```

Regularization Regularization penalizes records which stray far from the mean but have few associated ratings, such as an obscure film with only a few very low ratings. Following the derivation in the course, we can select the bias values using a regularization factor λ as follows:

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

$$\hat{b}_u(\lambda) = \frac{1}{\lambda + n_u} \sum_{u=1}^{n_u} \left(Y_{i,u,g} - \hat{b}_i - \hat{\mu}\right)$$

$$\hat{b}_g(\lambda) = \frac{1}{\lambda + n_g} \sum_{g=1}^{n_g} \left(Y_{i,u,g} - \hat{b}_i - \hat{b}_u - \hat{\mu}\right)$$

$$\hat{b}_y(\lambda) = \frac{1}{\lambda + n_y} \sum_{y=1}^{n_y} \left(Y_{i,u,g,y} - \hat{b}_i - \hat{b}_u - \hat{b}_g - \hat{\mu}\right)$$

$$\hat{b}_s(\lambda) = \frac{1}{\lambda + n_s} \sum_{s=1}^{n_s} \left(Y_{i,u,g,y,s} - \hat{b}_i - \hat{b}_u - \hat{b}_g - \hat{b_y} - \hat{\mu}\right)$$

A key step for regularization is the λ parameter estimation. We'll do it iterating values to obtain a minimum. For this estimation it is necessary **we do not use and test data**, to prevent overfitting and based in a basic machine learning rule. So we need to separate two sub-partitions (`train_r2` and `test_r2`) from the general training set. Due to the relative precision our iterative approach offers, we do not need to use all the data set, we can reduce it and improve computability.

```
# ==============
# Let's separate our test dataset to a train+test to evaluate some parameters only against training
# ==============
devReduction <- 0.02 # Percentage of original data we extract for development
set.seed(1, sample.kind="Rounding")
reduced_index <- createDataPartition(y = edx$rating, times = 1, p = devReduction, list = FALSE)
reduced_edx <- edx[reduced_index,]

devSplit <- 0.2 # partition train::test
set.seed(1, sample.kind="Rounding")
reduced2_index <- createDataPartition(y = reduced_edx$rating, times = 1, p = devSplit, list = FALSE)

edx_r2 <- reduced_edx[-reduced2_index,]
temp <- reduced_edx[reduced2_index,]

# Make sure userId and movieId in validation set are also in edx set
validation_r2 <- temp %>%
  semi_join(edx_r2, by = "movieId") %>%
  semi_join(edx_r2, by = "userId")
# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation_r2)
```

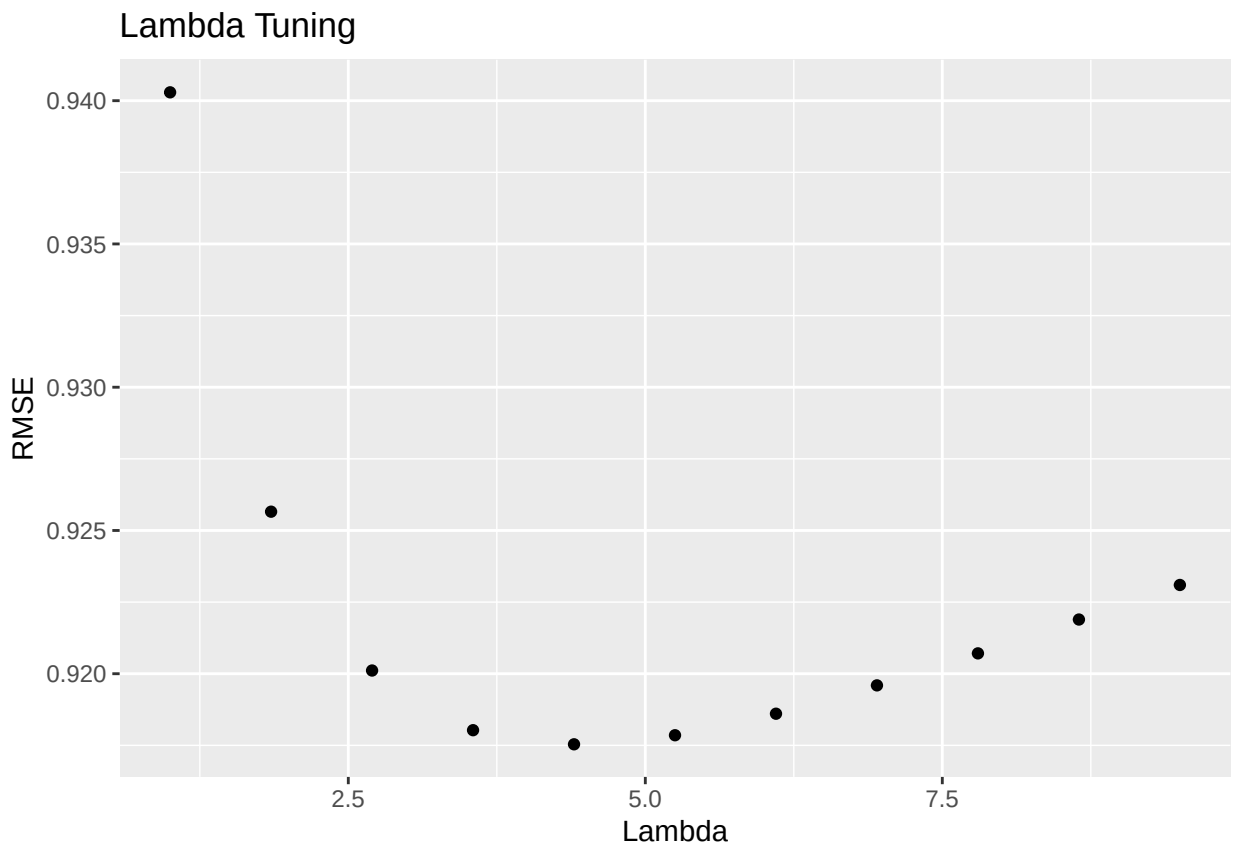
```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres", "year")
```

```
edx_r2 <- rbind(edx_r2, removed)
rm(reduced_edx, reduced_index, reduced2_index, temp, removed, devReduction, devSplit)
```

Now we are ready to estimate best λ to minimize RMSE in our reduced dataset.

```
lambdas <- seq(1, 10, 0.85)

lRMSEs <- sapply(lambdas, function(l){
  b_i <- edx_r2 %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- edx_r2 %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  b_g <- edx_r2 %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))

  b_y <- edx_r2 %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - b_i - b_u - b_g - mu)/(n()+l))

  predicted_ratings <- validation_r2 %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by="genres") %>%
    left_join(b_y, by = "year") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_y)

  return(RMSE(validation_r2$rating, predicted_ratings$pred))
})
```

This RMSE result is not stored and it is not significant. It is only used to estimate best lambda. We see how a good λ improves RMSE:




```
## [1] 4.4
```

Training and prediction We have now our estimation for best λ, so we are ready to train our model. The idea is to apply all precalculated effects per category to each observation to predict, using μ (global ratings average) as base for the correction.

Note that for computability, and taking into account that difference will be small, we will lightly simplify our model by:

1) using alphabetical genres lists of each movie instead of the bias of combination of specific genres, and
2) use the BING lexicon for assessing sentiments derived from movie's title, since this lexicon assigns only positive/negative instead of a longer set – these two sentiments covers and are implicitly linked to the longer list of emotions (in general converting each one to just positive or negative)

```
# ==========
# Training
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
```

```
b_g <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+lambda))

b_y <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - b_i - b_u - b_g - mu)/(n()+lambda))

b_sr <- (left_join(edx, byMovie_SentimentEffect, by="movieId")) %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_y, by="year") %>%
  group_by(movieId) %>%
  summarize(b_sr = sum(rating - b_i - b_u - b_g - b_y - mu)/(n()+lambda))

# ==========
# Prediction
predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_y, by = "year") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_y)

# ==========
# Calculate and store RMSE
mean_umgR_RMSE <- RMSE(validation$rating, predicted_ratings$pred)
mean_results <- bind_rows(mean_results, data_frame(Method="Phase 1 (Regularized Mov+Usr+Gnr+Yr)", RMSE =
```

We have obtained a much better RMSE that just the average, or that the approach just user-movie shown in the course:

Method	RMSE
Mean as base reference	1.0612
Phase 1 (Regularized Mov+Usr+Gnr+Yr)	0.8643

Phase 2 Model

We will use predictions obtained in Phase 1 Model as base reference, instead of μ, for discounting other biases: weekday, month, movie-age-at-rating. We need to prepare an extended version of the train and test datasets to have new features available:

```
# Adding predicted rating (in Phase 1) and calculated time-related prescriptors
# (using timestamp as input)
edx_ph2 <- edx %>%
```

```
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_y, by = "year") %>%
  mutate(WeekDay = factor(weekdays(as_datetime(edx$timestamp))),
         Month = month(as_datetime(timestamp)),
         AgeAtRating = year(as_datetime(timestamp)) - year,
         unbiasedRating = mu + b_i + b_u + b_g + b_y)
```

And also need to extend predicted ratings from previous model with new features

```
# Adding also to test set
validation_ph2 <- predicted_ratings %>%
  mutate(WeekDay = factor(weekdays(as_datetime(timestamp))),
         Month = month(as_datetime(timestamp)),
         AgeAtRating = year(as_datetime(timestamp)) - year)
```

We see regularization will not drive us to significant changes: as we visualized during data exploration, biases are so small and distribution of ratings per category so homogeneous for new prescriptors (WeekDay, Month, AgeAtRating), that it is not worthy to regularize. Instead, we´ll use **lambda2** as zero, though including parameter in the code for further developments if applicable.

```
lambda2 <- 0
```

We are now ready to train and predict our model

```
# Training Phase 2 model
b_w <- edx_ph2 %>%
  group_by(WeekDay) %>%
  summarize(b_w = sum(rating - unbiasedRating)/(n()+lambda2))

b_m <- edx_ph2 %>%
  left_join(b_w, by="WeekDay") %>%
  group_by(Month) %>%
  summarize(b_m = sum(rating - b_w - unbiasedRating)/(n()+lambda2))

b_a <- edx_ph2 %>%
  left_join(b_w, by="WeekDay") %>%
  left_join(b_m, by="Month") %>%
  group_by(AgeAtRating) %>%
  summarize(b_a = sum(rating - b_w - b_m - unbiasedRating)/(n()+lambda2))

# Prediction Phase 2 model
# We add to initial predictions rating-time information
validation_ph2 <- validation_ph2 %>%
  left_join(b_w, by = "WeekDay") %>%
  left_join(b_m, by = "Month") %>%
  left_join(b_a, by = "AgeAtRating") %>% mutate(pred = pred + b_w + b_m + b_a)

# Calculate and store RMSE
mean_rMUGYT_RMSE <- RMSE(validation$rating, validation_ph2$pred)
mean_results <- bind_rows(mean_results, data_frame(Method="Phase 2 (MUGY + TIME)", RMSE = mean_rMUGYT_RM
```

We have finally ended our modelling obtaining a much better RMSE.

4 Results

We have got aa appretiable result just with Phase 1 Model, and in case we have event time data available for cases to predict, we will obtain even more improvement on RMSE. As a results sumamry, running the final algorithm of our modelling on the –validation– set yields the following rating:

Method	RMSE
Mean as base reference	1.0612
Phase 1 (Regularized Mov+Usr+Gnr+Yr)	0.8643
Phase 2 (MUGY + TIME)	0.8639

5 Conclusion

We have implemented a model that starts in the usual application of movie and user effects on the observations average, but then adding some significant categories to refine predictions: Genre, Year and Sentiment extracted from the movie's title. After regularization of these biases, we reach and improved RMSE.

Then We added a contribution to the model by ensembling a second (sequential, optional) model that refines first model predictions, based on Week Day, Month when rating is registered, and years difference between rating and release year of the movie. This again improves RMSE without need of regularization in this second case.

Despite of the significant size of of the dataset for a local environment, it is also remarkable that this approach has no computability limitations, implementing a light approach in terms of performance requirements.

Finally, we can mention some potential improvements for future versions:

- Although impact is not very significant, there is a possibility to assess Genre-bias individually per Genre instead of as a set for each movie.
- A *content based filtering* based in matrices could be implemented for cases where we are predicting users with no previous ratings. However, given the size of dataset, this would probably mean a performance risk in a local environment.
- There is a possibility to explore *collaborative based filtering* methods implemented in package `RecommenderLab`: UBCF (User-Based Collaborative Filtering), IBCF (Item-Based Collaborative Filtering) or SVD (Singular Value Decomposition). Again, this approach is much more performance risk demanding.

As a final conclusion, we can state we have reached challenge goal in terms of project requirements and RMSE for the validation set provided.