Movielens Capstone project

Introduction

This is a personnal work about the movielens project. The objective is to propose a model to estimate movie ratings for a given set of users. The dataset is built as in the course instructions. A RMSE function is used to compare the models. A first model takes in consideration a time effect, as the ratings show a variable trend according to their time-stamp.

We are going to use the following libraries:

```
library(tidyverse)
library(caret)
library(lubridate)
library(gridExtra)
```

Preliminaries

Dataset

We first load the `edx` training and `validation` set. These were initialised from the code provided in the course material, by separating the global movielens dat in a 90/10 proportion.

```
load(file.path("rda","initial-set.rda"))
nrow(edx)
```

```
## [1] 9000055
```

```
nrow(validation)
```

```
## [1] 999999
```

Here is a subset of our data:

```
tibble(edx) %>% head
```

```
## # A tibble: 6 x 6
##   userId movieId rating timestamp title                  genres
##    <int>   <dbl>  <dbl>     <int> <chr>                  <chr>
## 1      1     122      5 838985046 Boomerang (1992)       Comedy|Romance
## 2      1     185      5 838983525 Net, The (1995)        Action|Crime|Thriller
## 3      1     292      5 838983421 Outbreak (1995)        Action|Drama|Sci-Fi|T~
## 4      1     316      5 838983392 Stargate (1994)        Action|Adventure|Sci-~
## 5      1     329      5 838983392 Star Trek: Generations~ Action|Adventure|Dram~
## 6      1     355      5 838984474 Flintstones, The (1994) Children|Comedy|Fanta~
```

The `edx` set contains 10677 movies and 69878 users. This is a large dataset and we must account for it in our modeling approach.

RMSE function

We define a RMSE function that will help us to compare the different models by calculating the mean square error, between the prediction vector and the vector of real ratings.

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

First model : time effect

As a first step we are checking for a time effect in ratings, then we'll compute a corrective term to account for this in the following steps.

Global average rating

First, let's calculate the global average rating.

```
mu <- mean(edx$rating)
mu
```

```
## [1] 3.512465
```

As a very basic model we consider the average rating mu as prediction. We evaluate this with the validation set.

```
predicted_ratings <- rep(mu,length(validation$rating))
rmse_val <- RMSE(predicted_ratings, validation$rating)
rmse_results <- tibble(method = "Prediction with the mean rating", RMSE = rmse_val)
rmse_val
```

```
## [1] 1.061202
```

Ratings evolution through time

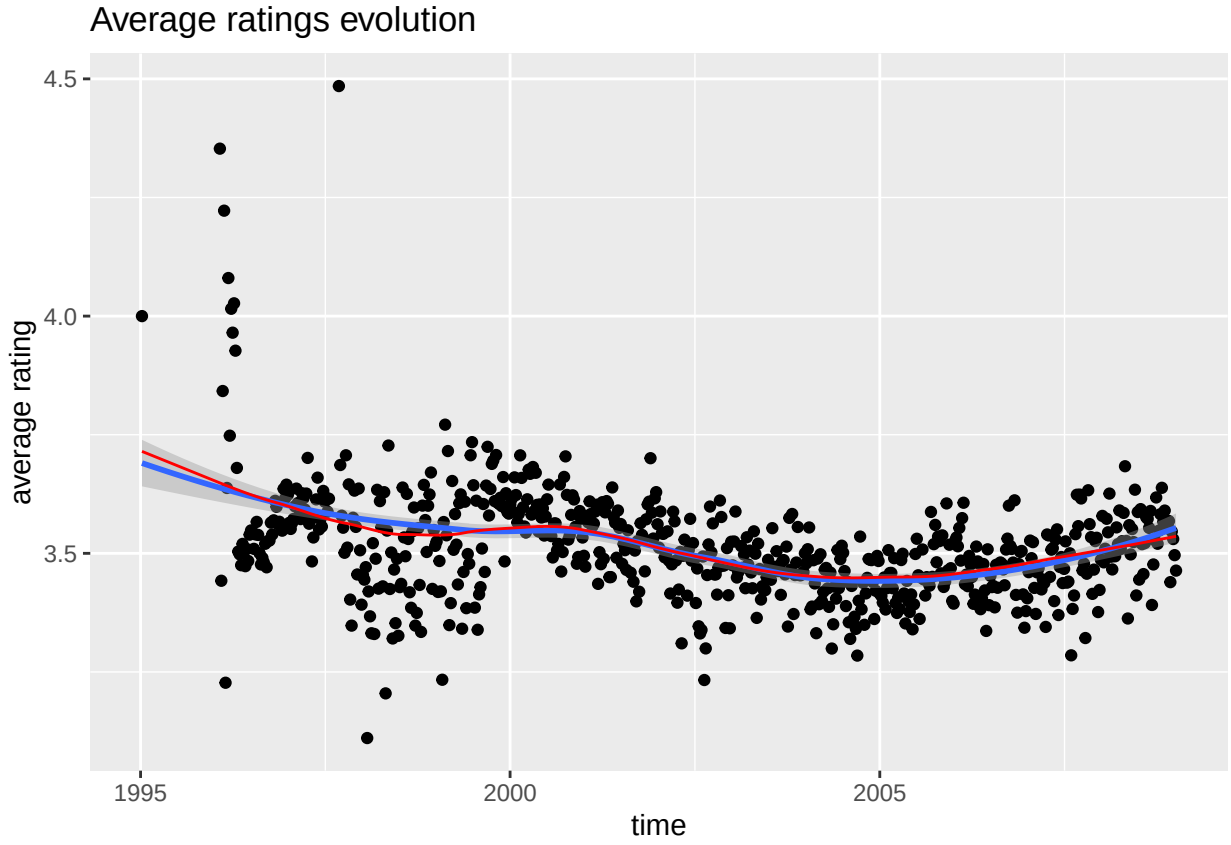
We consider the time effect with a one week granularity. We compute the difference between the current mean and the long term mean (mu). Then we fit a loess model in order to smooth the time effect. We set the span to 5 years.

```
week_effect <- edx %>%
  mutate (date = as_datetime(timestamp), week=round_date(date,unit="week")) %>%
  group_by(week) %>%
  summarize(avgweekeffect = mean(rating)-mu)
span <- 5*52/nrow(week_effect)
weekfit<-loess(avgweekeffect~as.numeric(week),degree = 1, span = span, data = week_effect)
```

This plot shows the weekly average rating (black dots), with the associated geom_smooth in blue, and our loess time effect in red. The red and blue lines are very close, which we wanted.

```
edx %>% mutate (date = as_datetime(timestamp), week=round_date(date,unit="week")) %>%
  group_by(week) %>%
  summarize(weekrating = mean(rating)) %>%
  mutate(timeeffect = mu+weekfit$fitted) %>%
  ggplot(aes(week, weekrating)) +
  geom_point() +  geom_smooth() +
  geom_line(aes(week, timeeffect), color="red") +
  xlab("time")+ylab("average rating")+ggtitle("Average ratings evolution")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Time-effect model

Our model is:
$Y = \mu + TE(t)$
where mu is the long term average, TE is the time effect function of time t.
We evaluate the time-effect model against the validation set. Note that there is hopefully no missing time-effect point for the merge with the validation set, else we would have needed to interpolate the missing values.

```
time_effect <- data.frame(week=week_effect$week, time_effect = weekfit$fitted)

predicted_ratings <- validation %>%
  mutate (date = as_datetime(timestamp), week=round_date(date,unit="week")) %>%
  left_join(time_effect, by="week") %>%
  mutate(pred = mu + time_effect ) %>%
  pull(pred)

rmse_val <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Additional Time effect", RMSE = rmse_val ))
rmse_results %>% knitr::kable()
```

method	RMSE
Prediction with the mean rating	1.061202
Additional Time effect	1.060070

If we compare the RMSE, we can see that the time-effect brings a very small improvement to our predictions. We're going to look at other additional effects. We keep the time effect to the `edx` set for future use.

```
edx <- edx %>%
  mutate (date = as_datetime(timestamp), week=round_date(date,unit="week")) %>%
  left_join(time_effect, by="week")
```

Second model : user and movie effect

As in the course material, we consider the model :
$Y(u,i) = \mu + TE(t) + b_u + b_i + \epsilon(u,i)$

Where:
$Y(u,i)$ is the rating of movie i by user u
$TE(t)$ is the time effect for time t
b_u is the effect for user u
b_i is the effect for movie i
$\epsilon(u,i)$ are independant random variables of mean 0

Regularization

We use a regularization factor `lambda` to minimize the total variance
$V = \sum(\hat{y} - Y)^2 + lambda * (\sum b_i^2 + \sum b_u^2)$

We partition the training set `edx` in order to look for an optimal `lambda` with a 20% partition. We make sure the test set does not contain unknown items.

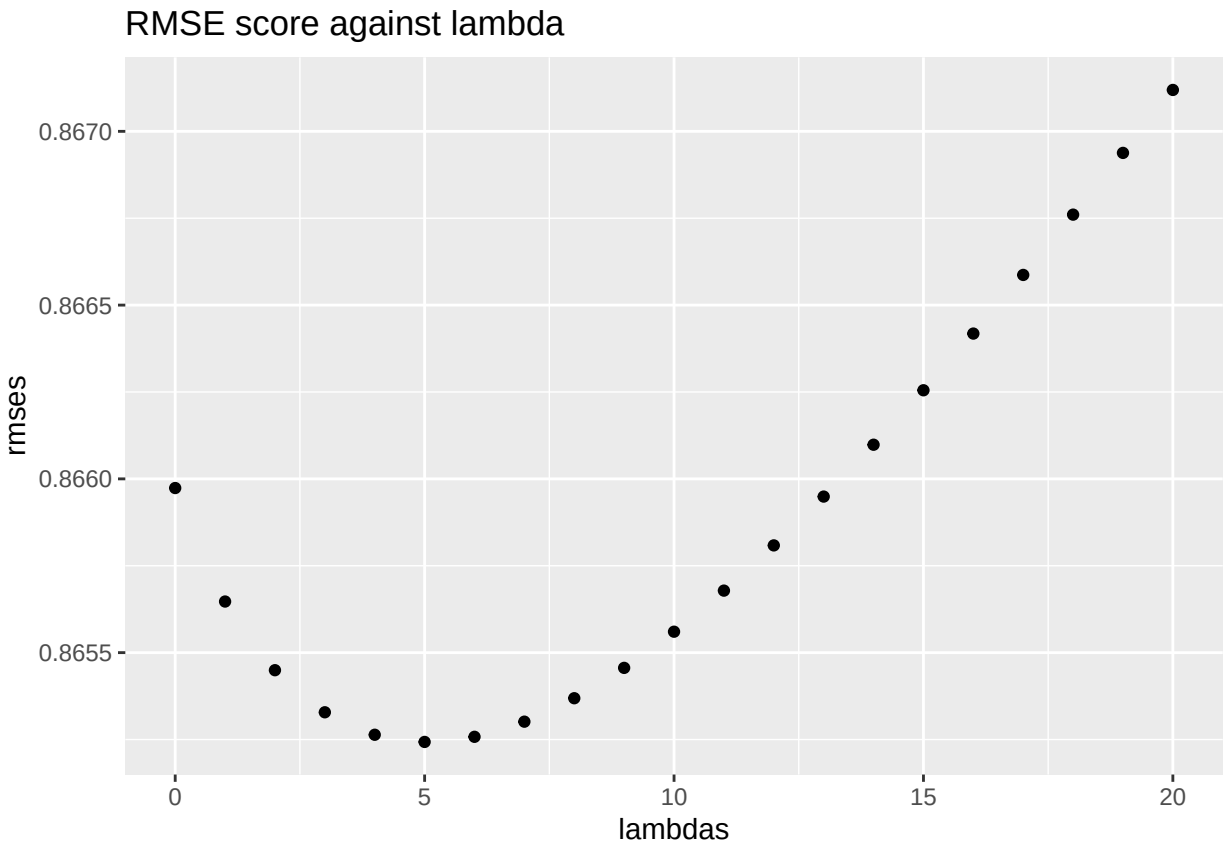
```
edx_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_lambda <- edx[-edx_index,]
temp <- edx[edx_index,]

test_lambda <- temp %>%
  semi_join(train_lambda, by = "movieId") %>%
  semi_join(train_lambda, by = "userId")
removed <- anti_join(temp, test_lambda, by = c("userId", "movieId"))
train_lambda <- rbind(train_lambda, removed)
```

We look for a lambda with the lower RMSE. Our training set is `train_lambda` and our testing set is `test_lambda`.

```
lambdas <- seq(0, 20, 1)
rmses <- sapply(lambdas, function(l){
  b_i <- train_lambda %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu - time_effect)/(n()+l))
  b_u <- train_lambda %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu - time_effect)/(n()+l))
  predicted_ratings <-
    test_lambda %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + time_effect + b_i + b_u) %>%
    pull(pred)
```

```
  return(RMSE(predicted_ratings, test_lambda$rating))
})
```

```
qplot(lambdas, rmses, main="RMSE score against lambda")
```




```
lambda <- lambdas[which.min(rmses)]
lambda
```

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

We'll keep the value of lambda=5 for the future steps. Now we compute the b_u and b_i vectors with the final value of lambda.

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

Score

To assess our model we compute the RMSE score against the validation set, using the model vectors calibrated over the training set.

```
predicted_ratings <- validation %>%
  mutate (date = as_datetime(timestamp), week=round_date(date,unit="week")) %>%
  left_join(time_effect, by="week") %>%
```

```
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + time_effect + b_i + b_u) %>%
  pull(pred)

rmse_val <- RMSE(predicted_ratings, validation$rating)
rmse_val
```

```
## [1] 0.8642395
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="User and movie effects model", RMSE = rmse_val ))
rmse_results %>% knitr::kable()
```

method	RMSE
Prediction with the mean rating	1.0612018
Additional Time effect	1.0600697
User and movie effects model	0.8642395

Genre effect model

Exploring the residuals and breaking by genres

To improve the prediction, we try to explore the residuals broken by genre.
First we evaluate the whole list of atomic values for column genres:

```
genres_l <- edx %>%
  pull(genres) %>% unique()
genres_l <- str_split(genres_l,"\\|", simplify =  FALSE)
genres_l <- sapply(genres_l, function(x) {paste(x, sep=" ",collapse=" ") })
genres_l <- paste(genres_l, collapse=" ")
genres_l <- data.frame(str_split(genres_l," "))%>% unique
names(genres_l)<-c("genre")
genre_columns <- str_c("genre",seq(1,nrow(genres_l)),sep="_")
```

This will help us to detail the ratings per unitary genres:

```
edx %>% filter(userId==1 & movieId==292) %>% select(userId,movieId,genres) %>% knitr::kable()
```

userId	movieId	genres
1	292	Action|Drama|Sci-Fi|Thriller

```
edx%>% filter(userId==1 & movieId==292) %>%
  separate(genres,sep="\\|", into = genre_columns, fill="right",extra="drop") %>%
  gather(dummy, genre, all_of(genre_columns)) %>%
  select(-dummy) %>%
  filter(!is.na(genre)) %>%
  select(userId,movieId,genre) %>% knitr::kable()
```

userId	movieId	genre
1	292	Action
1	292	Drama

userId	movieId	genre
1	292	Sci-Fi
1	292	Thriller

Memory consumption is important with the whole dataset. To ease things we'll focus on two subsets and explore movie genres inside thses two subsets :
* one subset containing the data related to the 100 users giving the worst predictions
* one subset containing the data for a random selection of 100 users

```
users_worst <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(residual = mu + b_i + b_u + time_effect - rating) %>%
  arrange(desc(abs(residual))) %>%
  select(userId)  %>%  unique %>% slice(1:100) %>% pull(userId)

users_rand <- edx %>%
  filter( userId %in% sample(userId, 100) ) %>%
  pull(userId) %>% unique

edx_worst <- edx %>% filter(userId %in% users_worst) %>%
  separate(genres,sep="\\|", into = genre_columns, fill="right",extra="drop") %>%
  gather(dummy, genre, all_of(genre_columns)) %>% select(-dummy) %>% filter(!is.na(genre))

edx_rand <- edx %>% filter(userId %in% users_rand) %>%
  separate(genres,sep="\\|", into = genre_columns, fill="right",extra="drop") %>%
  gather(dummy, genre, all_of(genre_columns)) %>% select(-dummy) %>% filter(!is.na(genre))
```

Let's have a look at the ratings for the two groups, broken by genre.
Worst group.

```
edx_worst %>% group_by(genre) %>%
  summarize(count=n(), mean=mean(rating), median=median(rating),sd=sd(rating)) %>%
  arrange (mean) %>% knitr::kable()
```

genre	count	mean	median	sd
Documentary	310	3.779032	4.5	1.571504
IMAX	30	3.900000	4.0	1.335200
Drama	8970	3.957915	4.5	1.317560
Horror	2398	3.965596	4.0	1.221233
Comedy	9999	3.967347	4.5	1.197266
Crime	2911	3.971487	4.5	1.238019
Musical	1421	3.971851	4.5	1.285996
Children	2874	3.979993	4.0	1.174491
War	1161	3.981482	4.5	1.340950
Mystery	1236	3.983010	4.5	1.246743
Animation	1472	3.985394	4.5	1.207820
Sci-Fi	3767	3.990178	4.5	1.197297
Thriller	5532	4.000452	4.5	1.204032
Fantasy	2824	4.004426	4.5	1.205166
Action	6984	4.023411	4.5	1.169014
Film-Noir	276	4.034420	4.5	1.310688
Romance	4141	4.035861	4.5	1.207157

genre	count	mean	median	sd
Adventure	5434	4.046099	4.5	1.168541
Western	528	4.130682	4.5	1.069220

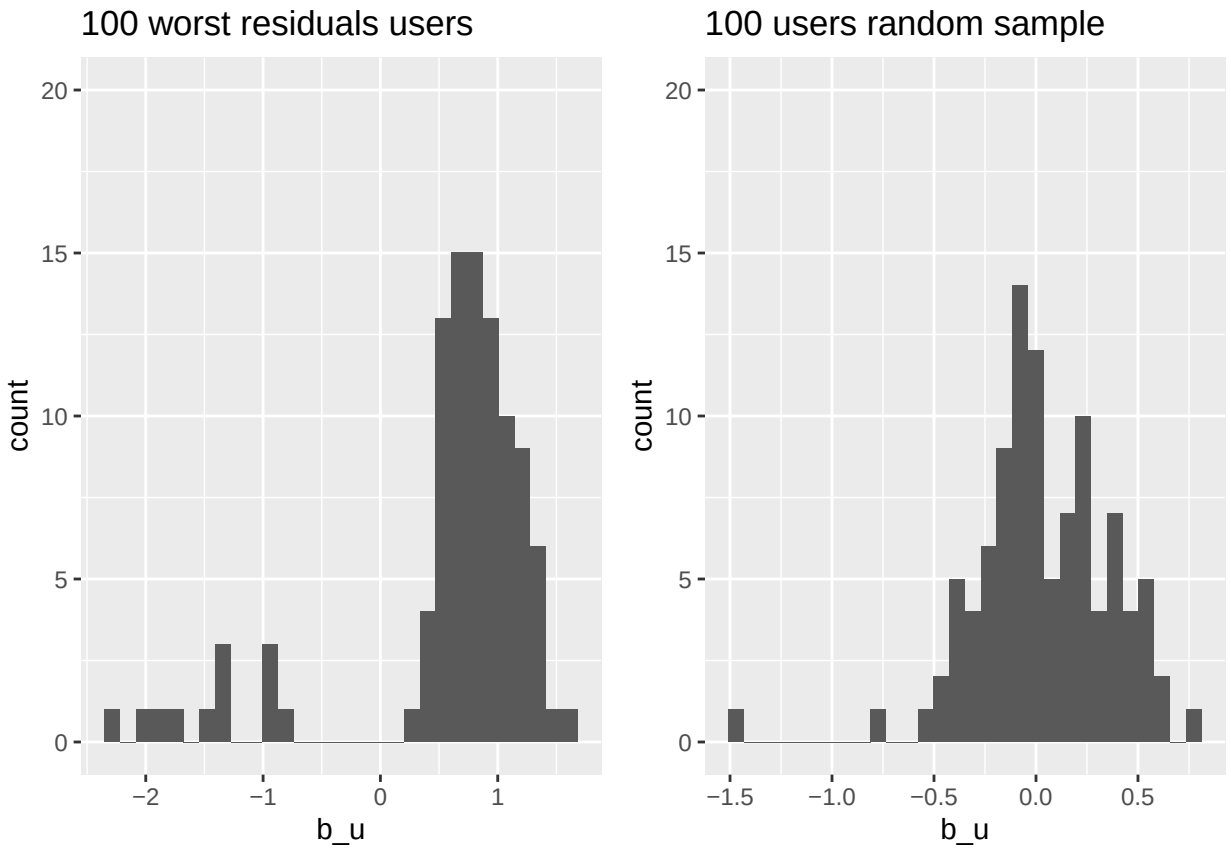
Random group.

```
edx_rand %>% group_by(genre) %>%
  summarize(count=n(), mean=mean(rating), median=median(rating),sd=sd(rating)) %>%
  arrange (mean)%>% knitr::kable()
```

genre	count	mean	median	sd
Horror	2679	3.265957	3.5	1.1564114
Sci-Fi	4444	3.293542	3.5	1.1489189
Children	2808	3.306268	3.5	1.0960482
Action	8522	3.308378	3.5	1.1217453
Adventure	6543	3.397753	3.5	1.0979329
Comedy	13613	3.401491	3.5	1.0867764
Fantasy	3410	3.412024	3.5	1.1229240
Thriller	8114	3.467032	3.5	1.0684067
Musical	1649	3.492420	3.5	1.0695349
Animation	1824	3.523300	4.0	1.0325901
IMAX	44	3.534091	3.5	1.0252494
Romance	6566	3.554371	4.0	1.0400648
Western	645	3.610853	4.0	1.0002515
Crime	4639	3.622009	4.0	1.0303128
Mystery	2188	3.684872	4.0	1.0064877
Drama	15516	3.700342	4.0	0.9753484
War	1786	3.704087	4.0	1.0354664
Documentary	422	3.892180	4.0	0.9309207
Film-Noir	502	4.026892	4.0	0.9348935

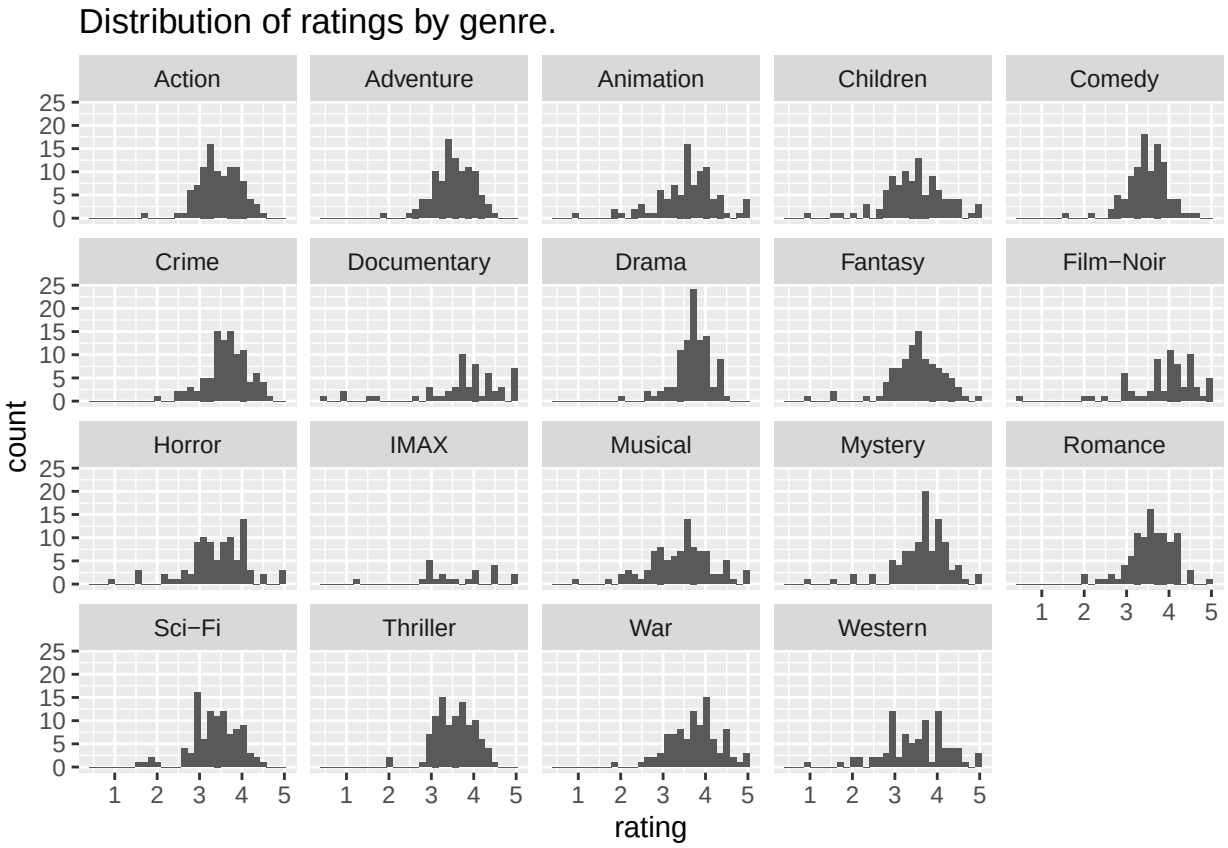
It appears that the worst group ratings are less close to the average μ than the random group. If we check at the user effect b_u we can see it is more important on the worst group. This could explain the amplification of residual errors.

```
h1 <- edx_worst %>%
  left_join(b_u) %>%
  group_by(userId) %>%
  summarize(b_u = first(b_u) ) %>%
  ggplot(aes(b_u)) + geom_histogram() +
  ggtitle("100 worst residuals users") + ylim(0,20)
h2 <- edx_rand %>%
  left_join(b_u) %>%
  group_by(userId) %>%
  summarize(b_u = first(b_u) ) %>%
  ggplot(aes(b_u)) + geom_histogram() +
  ggtitle("100 users random sample") + ylim(0,20)
grid.arrange(h1, h2, ncol = 2)
```



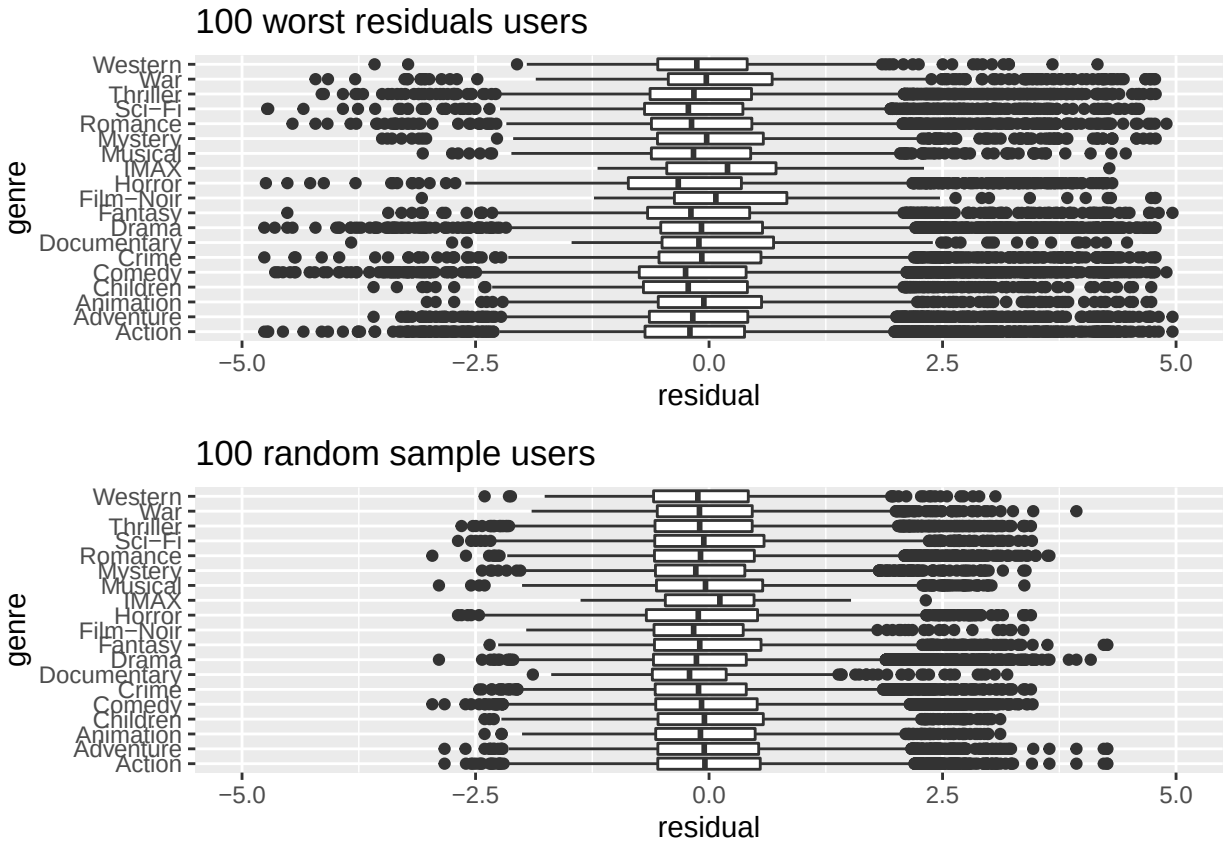
If we display the histograms of ratings, broken by genre, for the random selection of users, it shows clearly a general genre effect :

```
edx_rand %>%
  group_by(genre, userId) %>% summarize(rating=mean(rating)) %>%
  ggplot(aes(rating)) + geom_histogram() +
  facet_wrap(. ~ genre) + ggtitle("Distribution of ratings by genre.")
```



Now we look at the ratings boxplots broken by genre, and compare the two groups. We can see the variance is globally much more important in the in the group of worst rating users, and there is also more variance between the different genres for that group.

```
h1 <- edx_worst  %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(residual = mu + b_i + b_u + time_effect - rating) %>%
  ggplot(aes(y=genre,x=residual))+geom_boxplot()+ggtitle("100 worst residuals users")+xlim(-5,5)
h2 <- edx_rand  %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(residual = mu + b_i + b_u + time_effect - rating) %>%
  ggplot(aes(y=genre,x=residual))+geom_boxplot()+ggtitle("100 random sample users")+xlim(-5,5)
grid.arrange(h1, h2, nrow = 2)
```



In conclusion, there seems to be a genre effect. That effect could be approximately similar for a large part of users. For users with the worst predictions the genre effect seems to be different from one user to another. So we could try take into account a variability of genre for each user.

Modeling the genre effect

We assume a model accounting for a mean rating per genre for each user:

$$Y(i,u) = \mu + TE(t) + b_i + b_u + mean(b_{u,k})$$

where: $b_{u,k}$ is the average residual for user u and movies of genre k. The mean is calculated over the genres k related to movie i.

First we perform a tidy of the genres column, and create a movie/genre separate table to optimize memory consumption.

```
moviegenres <- edx %>% select(movieId,genres) %>% distinct() %>%
  separate(genres,sep="\\|",
           into = tidyselect::all_of(genre_columns),
           fill="right",
           extra="drop") %>%
  gather(dummy, genre, tidyselect::all_of(genre_columns)) %>% select(-dummy) %>%
  filter(!is.na(genre))

head(moviegenres)%>% knitr::kable()
```

movieId	genre
122	Comedy
185	Action
292	Action
316	Action
329	Action
355	Children

Now let's compute the new predictor, b_u_k, for each user and genre. Note that regularisation here seems to bring no benefit.

```
b_u_k <- edx %>%
  left_join(moviegenres, by="movieId") %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(residual = mu + b_i + b_u + time_effect - rating) %>%
  group_by(userId, genre) %>%
  summarize(b_u_k = sum(rating -time_effect - b_i - b_u -mu)/n())
```

To illustrate, let's consider an example: user id 1 with movie id 122.

```
edx %>% filter(userId==1 & movieId==122) %>% knitr::kable()
```

userId	movieId	rating	timestamp	title	genres	date	week	time
1	122	5	838985046	Boomerang (1992)	Comedy|Romance	1996-08-02 11:24:06	1996-08-04	0.1

```
ex_bi<-b_i %>% filter(movieId==122) %>% pull(b_i)
ex_bu<-b_u %>% filter(userId==1) %>% pull(b_u)
ex_te <- edx %>% filter(userId==1 & movieId==122) %>% pull(time_effect)
ex_buk <- b_u_k %>% filter(userId==1 & genre %in% c("Comedy","Romance"))
```

The user-genre contribution shows 2 lines for the considered movie

userId	genre	b_u_k
1	Comedy	0.5189817
1	Romance	0.1789609

We thus have a contribution of `mean(ex_buk$b_u_k)` = 0.3489713 for the genre effect.
The final prediction is :

```
tibble (mu=mu, b_u=ex_bu, b_i=ex_bi,b_uki=mean(ex_buk$b_u_k), te=ex_te) %>%
  mutate(prediction = mu + b_u + b_i + te + b_uki)
```

```
## # A tibble: 1 x 6
##      mu   b_u    b_i b_uki    te prediction
##   <dbl> <dbl>  <dbl> <dbl> <dbl>      <dbl>
## 1  3.51  1.28 -0.710 0.349 0.106       4.54
```

RMSE Score

We compute the RMSE on the validation set. Note that it can happen that a combination of user+genre was not present in the training set. In that case the effect will be null for the given genre. We also cap the

prediction to a value of 5 and floor it to 1.

```
pred_val <- validation %>%
  mutate (date = as_datetime(timestamp), week=round_date(date,unit="week")) %>%
  left_join(time_effect, by="week") %>%
  left_join(moviegenres, by = "movieId") %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_u_k, by = c("userId","genre")) %>%
  group_by(userId,movieId) %>%
  summarize(b_uki = mean(b_u_k, na.rm = TRUE),
            b_u = first(b_u),
            b_i = first(b_i),
            rating = first(rating),
            time_effect = first(time_effect)) %>%
  mutate( estimate = mu + time_effect + b_i + b_u + ifelse(!is.na(b_uki), b_uki,0),
          pred = ifelse(estimate > 5,5, ifelse(estimate<1,1,estimate))) %>%
  select(userId,movieId,pred)

predicted_val <- left_join(validation, pred_val , by=c("userId","movieId")) %>% pull(pred)
rmse_val <- RMSE(predicted_val, validation$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Additional User-Genre effect model", RMSE = rmse_val ))
rmse_results %>% knitr::kable()
```

method	RMSE
Prediction with the mean rating	1.0612018
Additional Time effect	1.0600697
User and movie effects model	0.8642395
Additional User-Genre effect model	0.8488687

Other approaches

Here are a few methodologies I tried to explore. Since the results were not satisfactory I do not report these sections in details.

residual matrix factorization

The SVD factorization could be an interesting approach to catch the correlations between combinations of users and movies. We start from the user-movie model. We have to build a matrix of 70000 users x 10000 movies, containing the residuals (=prediction-rating). The NAs are replaced with zeros.
Unfortunately if we perform a SVD on the whole matrix we get short of memory on a 16GB RAM computer. Therefore, we limit the scope of the matrix on a subset of 6000 users and 1000 movies containing the worst predictions. On the 3 SVD matrices, we keep 500 first components out of 1000 in order to preserve around 90% of the total variance. Then we compute back the original matrix, which we can use to estimate a residual correction on the validation set.
The gain is approximately 0.0005 on the RMSE.

clustering

The clustering methods were applied on the user + movies + genre effect residual model.

Hierarchical

I used again the matrix of residuals, NA values filled with zero, and explored a hierarchical clustering on users. I tried different values for the cutoff height, but this was not succesfull to separate efficiently the users (I obtained a lot of groups of 1 user and a few very large groups).

Kmeans

The kmeans algorithms is applied (1) on the residual matrix, to compute groups of users, or (2) on the transposed matrix to compute groups of movies.
In each case, when inspecting the groups, we still have a large variance of residuals, thus it is not obvious how to exploit this information. Nevertheless I tried to use these groups with the random forest.

Trees

Random forest cannot be applied to a too large set of factors, so we have to exclude applying RF on every users or every movies. Rather, I tried to apply the RF on the groups of users and groups of movies that were computed using kmean. Unfortunately, though the idea seems interesting, the prediction is quite bad.

Conclusions

We finally have the following RMSE scores:

```
rmse_results %>% knitr::kable()
```

method	RMSE
Prediction with the mean rating	1.0612018
Additional Time effect	1.0600697
User and movie effects model	0.8642395
Additional User-Genre effect model	0.8488687

To improve significantly this score seems difficult without a specific model. The SVD factorisation can bring a small improvment. The clustering or regression trees methods might be analysed deeper to check for a possible improvment.