MovieLens Recommendation System

Dr. Ivan S. Zapreev

2020-01-03

According to Wikipedia:

> A recommender system or a recommendation system (sometimes replacing 'system' with a synonym such as platform or engine) is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item.

Introduction

Recommendation systems are widely used in commercial application to provide targeted commercials or suggest online content. In the latter, one can think of *Spotify* suggesting songs or *Youtube* proposing videos. The main goal of a recommendation system is to accurately predict client's preferences, based on available data on the previous user behavior. Note that, such data includes the behavior of all the clients.

The goal of this work is to build a movie recommendation system based on the MovieLens 10M dataset available from https://grouplens.org/datasets/movielens/10m/. To reach our goal we will use supervised machine learning techniques studied within the *"PH125.8x Data Science: Machine Learning"* course (a part of the broader HarvardX *Data Science Professional* certification program). The recommendation system will be therefore based on (linear) statistical models trained on the subset of the MovieLens 10M dataset.

The remainder of this section first present the data set, then defines the goal of this project more concretely, and finally identifies the main steps to be taken to reach the goal.

Dataset overview

As stated in the README of the original MovieLens 10M dataset, it contains `10000054` ratings and `95580` tags (not used in this study) applied to `10681` movies by `71567` users of the MovieLens service.

All ratings are contained in the file `ml-10M100K/ratings.dat`. Each line of this file represents one rating specified by the rating value (`rating`), the movie identifier (`movieId`), the identifier of the user (`userId`) and the submission timestamp (`timestamp`). Ratings are made on a `5`-star scale, with half-star increments. Timestamps represent seconds since midnight *Coordinated Universal Time (UTC) of January 1, 1970.*

Movie information is contained in the file `ml-10M100K/movies.dat`. Each line of this file represents one movie specified by its identifier (`movieId`), movie title (`title`) and the corresponding genres (`genres`). The movie titles are entered manually, so errors and inconsistencies may exist. Genres are pipe-separated lists, of the individual genres from the following set:

`("Action", "Adventure", "Animation", "Children's", "Comedy", "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror", "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western")`.

Note that, the MovieLens data set is not used "as is" but is pre-processed. The pre-processing steps will be described and explained in the *"Data preparation"* section of this document.

Project goal

The goal of the project is to build and train a statistical model predicting movie rating based on, at least, the user and the movie. The model is to be trained on the training set (`edx`) and is to be evaluation on a validation set (`validation`).[1]

The evaluation of model will be done using the residual mean squared error (RMSE) which, similarly to a standard deviation, can be interpreted as: *the typical error we make when predicting a movie rating.* In other words, e.g, if this number is larger than `1`, it means our typical error is larger than one star.

Definition *(RMSE)*

Let $y_{u,m}$ be the rating for movie m by user u and $\hat{y}_{u,m}$ be our prediction for that rating, then:

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

The definition above trivially translates into the following `R` function:

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

The ultimate goal of the project is to create a statistical model, solely based on the training set, that on the `validation` set will be able to predict the movie rating with $RMSE <= 0.8649$. The way we build such a model will be explained in the *"Modeling approach"* section of this document.

Execution plan

Let us now briefly outline the main steps to be performed to reach the previously formalized project goal:

1. **Prepare the data** – see the *"Data preparation"* section:
   - Select the relevant data; split into training and validation sets; and etc.
2. **Analyze the dataset** – see the *"Dataset analysis"* section:
   - Perform data exploration and visualization; summarize insights on the data.
3. **Describe the modeling approach** – see the *"Modeling approach"* section:
   - Consider the insights of the data analysis; suggest the way for building the prediction model.
4. **Present modeling results** – see the *"Results"* section:
   - Train the model on the `edx` set; analyze the training results; evaluate on the `validation` set.
5. **Provide concluding remarks** – see the *"Conclusions"* section:
   - Summarize the results; mention any approach limitations; outline possible future improvements.

Please note that, the remainder of the document is structured according to the execution plan listed above.

Data preparation

In this project, we use a pre-processed subset of the original dataset. Let us now explain how that subset obtained and pre-processed. Here we only use the data from the `ratings.dat` and `movies.dat` files from the MovieLens 10M dataset. To facilitate supervised learning, the data is pre-processed and split into a training (`edx`) and testing (`validation`) sets. The former will be used for training statistical model(s) and the latter will be used for the model(s) validation.

We pre-process and split the data in the following way:

[1] These sets are defined in the *"Data preparation"* section.

1. The `ratings.dat` and `movies.dat` are loaded into `ratings` and `movies` data frames
    1. `ratings` – with columns named userId, movieId, rating, timestamp
    2. `movies` – columns named movieId, title, genres
2. The ratings are coupled with the movies information by the `movieId` into a new `movielens` data frame
3. The `movielens` data frame is randomly split into two parts:
    2. `edx` – a set storing the % of the observations from the `movielens`, to be used for training
    3. `validation` – a set storing the % of the observations from the `movielens`, to be used for validation
4. The `validation` set is filtered to only contain movies and users present in `edx`
5. The observations filtered out in the previous step are added back to the `edx` set

For more details, see the `create_movielens_sets` function located in the `movielens_project.R` script. As a result of pre-processing we have two sets with the following metrics, the training set `edx`:

```
## # A tibble: 1 x 3
##   num_observations num_movies num_users
##              <int>      <int>     <int>
## 1          9000055      10677     69878
```

and the validation set `validation`:

```
## # A tibble: 1 x 3
##   num_observations num_movies num_users
##              <int>      <int>     <int>
## 1           999999       9809     68534
```

As one can see the `edx` set stores 90% of the total number of observations (`num_observations`) and the `validation` set contains 10%. The difference in the number of distinct movies (`num_movies`) and the number of distinct users (`num_users`) is explained by the observations being moved back from `validation` to `edx` in the last data preprocessing step, described above.

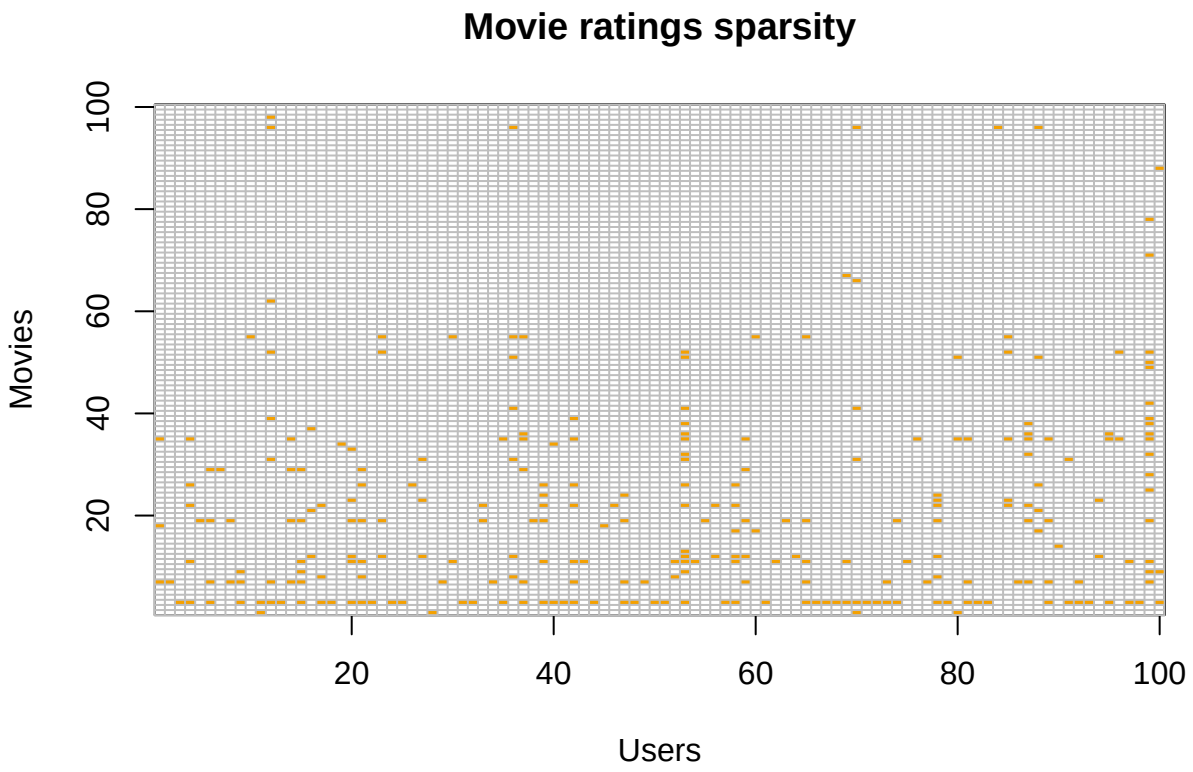
Given the training and validation set preparation procedure above we assume that the resulting sets follow the same probability distribution of the conditional probability of interest, see the section on the *"Modeling approach"*.

At last, let us mention that the `edx` and `validation` set analysis revealed no `N/A` values are present and also there was no need detected to do any additional cleaning of the data.

Dataset analysis

In this sections we analyze the training (`edx`) subset of the original MovieLens data set only, as we assume that this is the only data available for building the prediction models. The motivation behind is that we want to avoid influencing the model validation results. This is done under a, not-verified, assumption that the the testing set, follows the same probability distribution of the conditional probability of interest as the validation set. In the *"Results"* section we may come back to this assumption in case the accuracy of the developed statistical model(s) on the validation set will turn out to be insufficient.

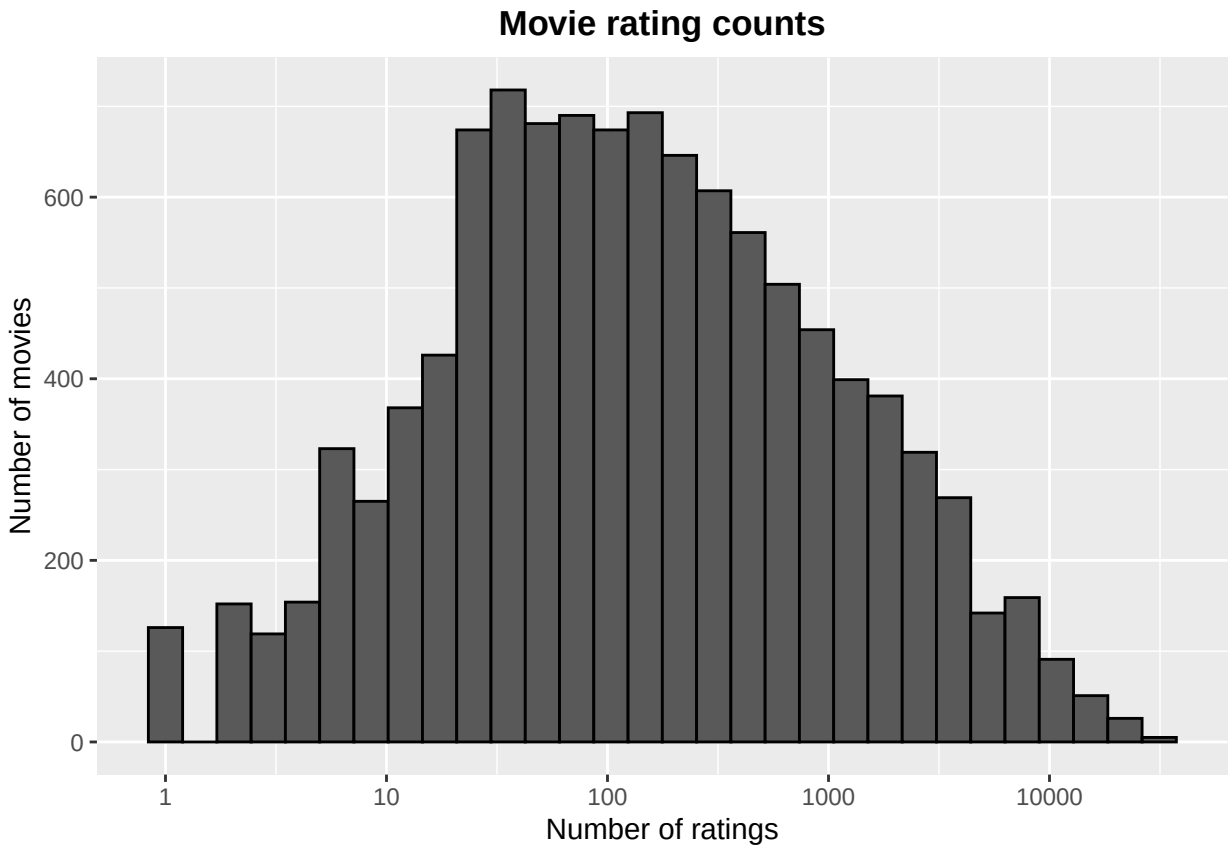
From the number of distinct user, movies and observations (ratings) in the `edx` set it is clear that not every user has rated every movie as $69878 * 10677 < 9000055$. The same can be seen via the next plot for randomly sampled `100` unique users and `100` unique movies:



The the task of a recommendation system we are to build can be seen as filling in the missing ratings. To stress the complexity of the undertaking, below we list a number of different observations that can influence rating predictions, and may need to be taken into account when building the statistical model.

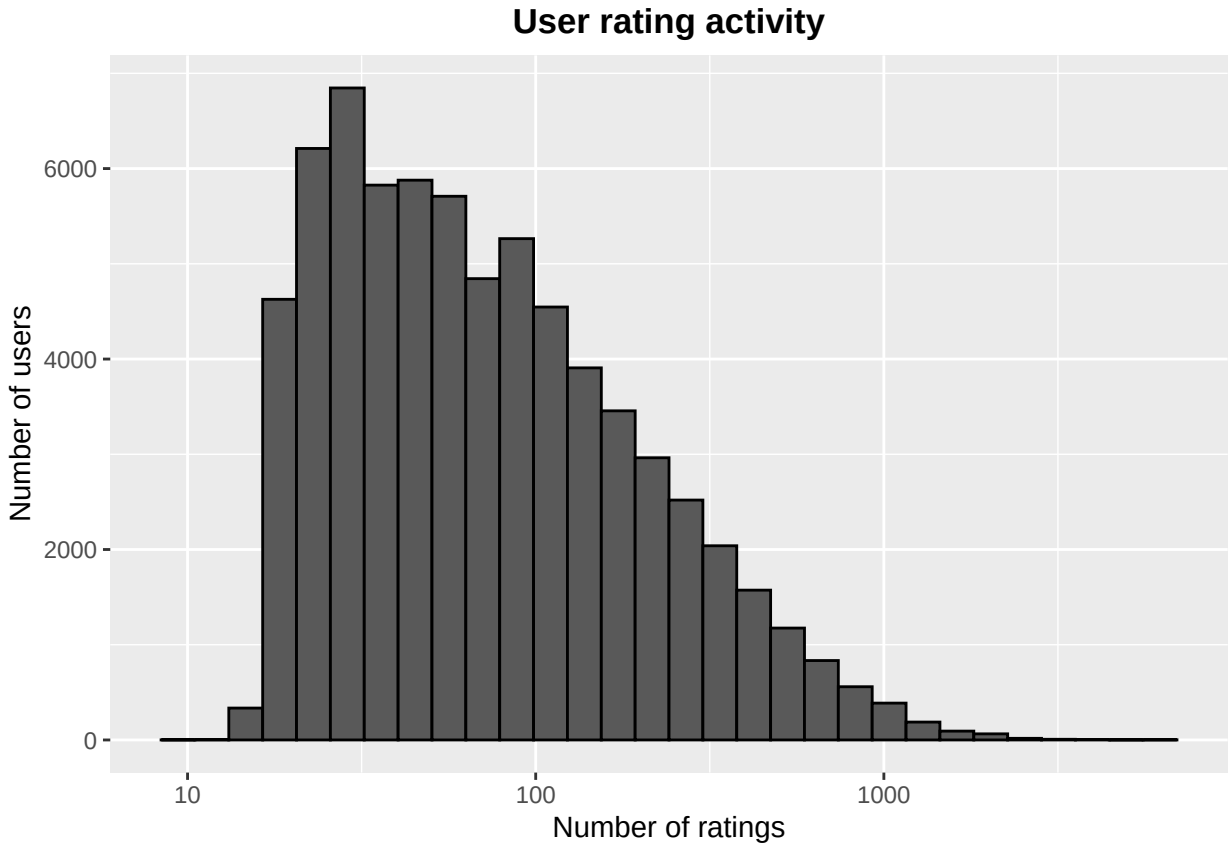
(O.I): Some movies are rated more often than the others:

```
movielens_data$edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  labs(x = "Number of ratings", y = "Number of movies") +
  ggtitle("Movie rating counts") +
  theme(plot.title = element_text(hjust = 0.5, face="bold"))
```



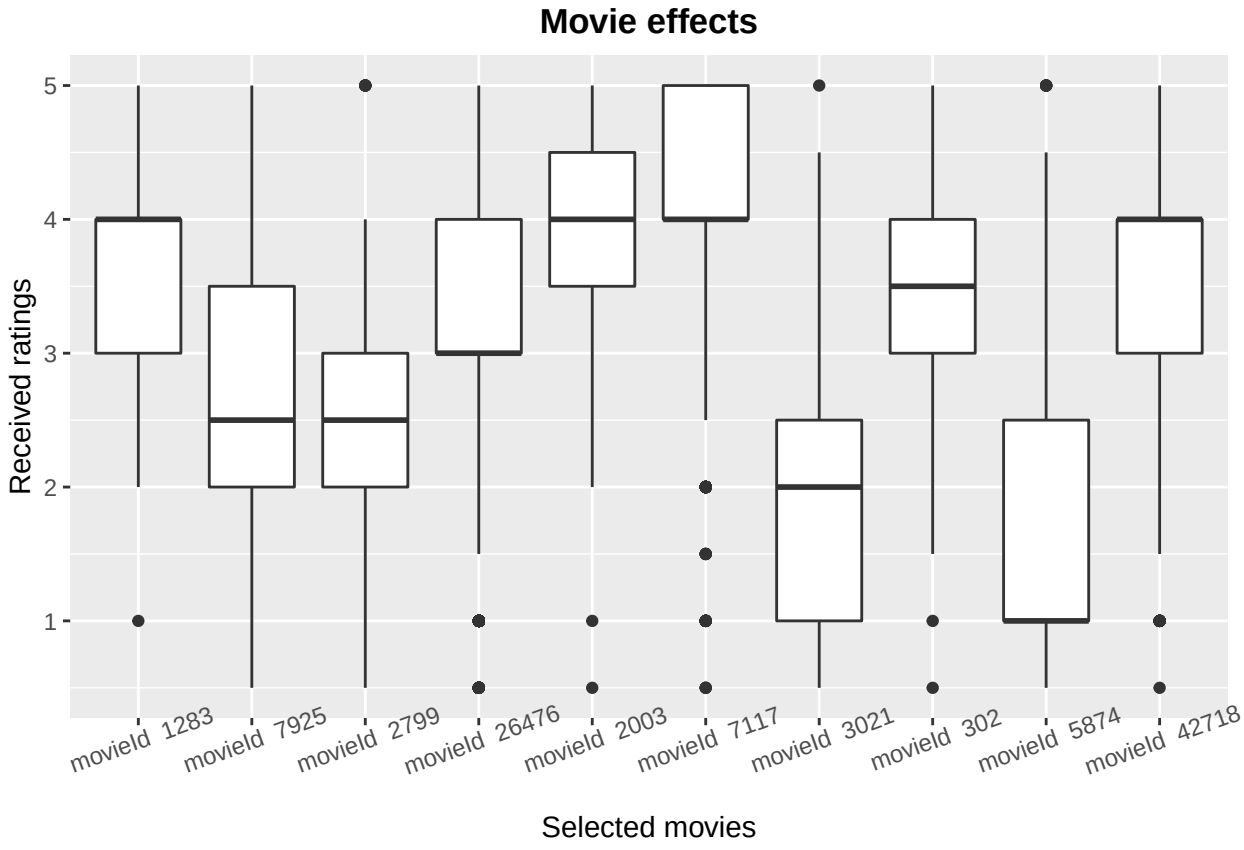
(O.II): Some users are more active in rating movies that the others:

```
movielens_data$edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  labs(x = "Number of ratings", y = "Number of users") +
  ggtitle("User rating activity") +
  theme(plot.title = element_text(hjust = 0.5, face="bold"))
```



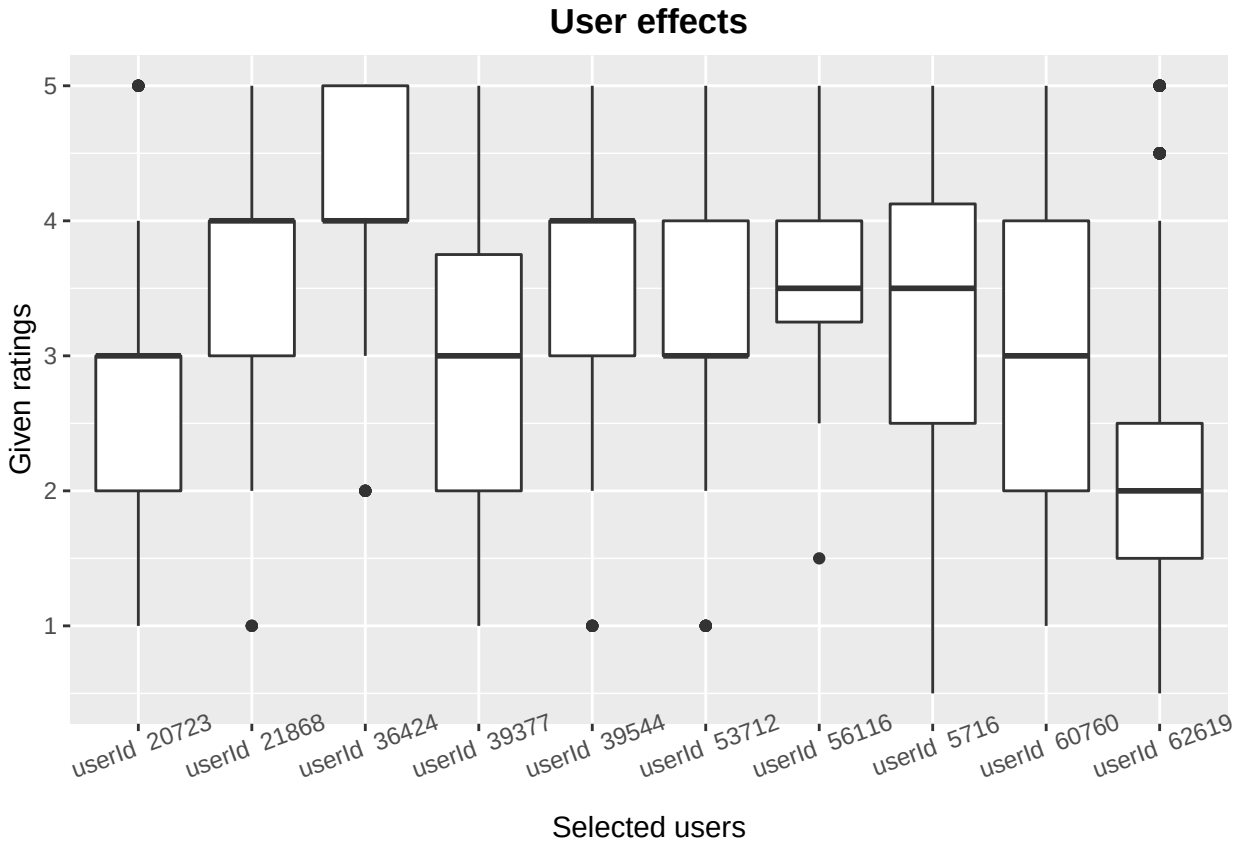
(O.III): Movies differ from each other in given ratings, lets consider 10 movies with at least 100 ratings each:

```
#Sample 10 individual movie ids for the movies with more than 100 ratings
movie_ids <- movielens_data$edx %>%
  group_by(movieId) %>% summarise(cnt = n()) %>%
  filter(cnt > 100) %>% pull(movieId) %>% unique(.)
set.seed(1)
sample_movie_ids <- sample(movie_ids, 10)

#Use the box plot to show variations in ratings between different movies
movielens_data$edx %>%
  filter(movieId %in% sample_movie_ids) %>%
  ggplot(aes(title, rating, group=movieId)) +
  geom_boxplot() +
  scale_x_discrete(labels = paste("movieId ",as.character(sample_movie_ids)))+
  theme(axis.text.x = element_text(angle = 20)) +
  labs(x = "Selected movies", y="Received ratings") +
  ggtitle("Movie effects") +
  theme(plot.title = element_text(hjust = 0.5, face="bold"))
```



(O.IV): Users differ from each other in giving ratings, lets consider 10 users with at least 100 ratings each:

```
#Sample 10 individual user ids for the users with more than 100 ratings
user_ids <- movielens_data$edx %>%
  group_by(userId) %>% summarise(cnt = n()) %>%
  filter(cnt > 100) %>% pull(userId) %>% unique(.)
set.seed(5)
sample_user_ids <- sample(user_ids, 10)

#Use the boxplot to show variations in ratings between different users
movielens_data$edx %>%
  filter(userId %in% sample_user_ids) %>%
  mutate(userName = paste("userId ", as.character(userId))) %>%
  group_by(userId) %>%
  ggplot(aes(userName, rating, group=userId)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 20)) +
  labs(x = "Selected users", y= "Given ratings") +
  ggtitle("User effects") +
  theme(plot.title = element_text(hjust = 0.5, face="bold"))
```



(O.V): The best and the worst (on-average) movies were not rated very often:

```
#Get the movies with the top 10 average scores
true_top_10 <- avg_movie_ratings %>%
  arrange(desc(avg_rating)) %>%
  slice(1:10)
true_top_10
```

```
## # A tibble: 10 x 3
##    movieId avg_rating num_ratings
##      <dbl>      <dbl>       <int>
##  1    3226       5             1
##  2   33264       5             2
##  3   42783       5             1
##  4   51209       5             1
##  5   53355       5             1
##  6   64275       5             1
##  7    5194       4.75          4
##  8   26048       4.75          4
##  9   26073       4.75          4
## 10   65001       4.75          2
```

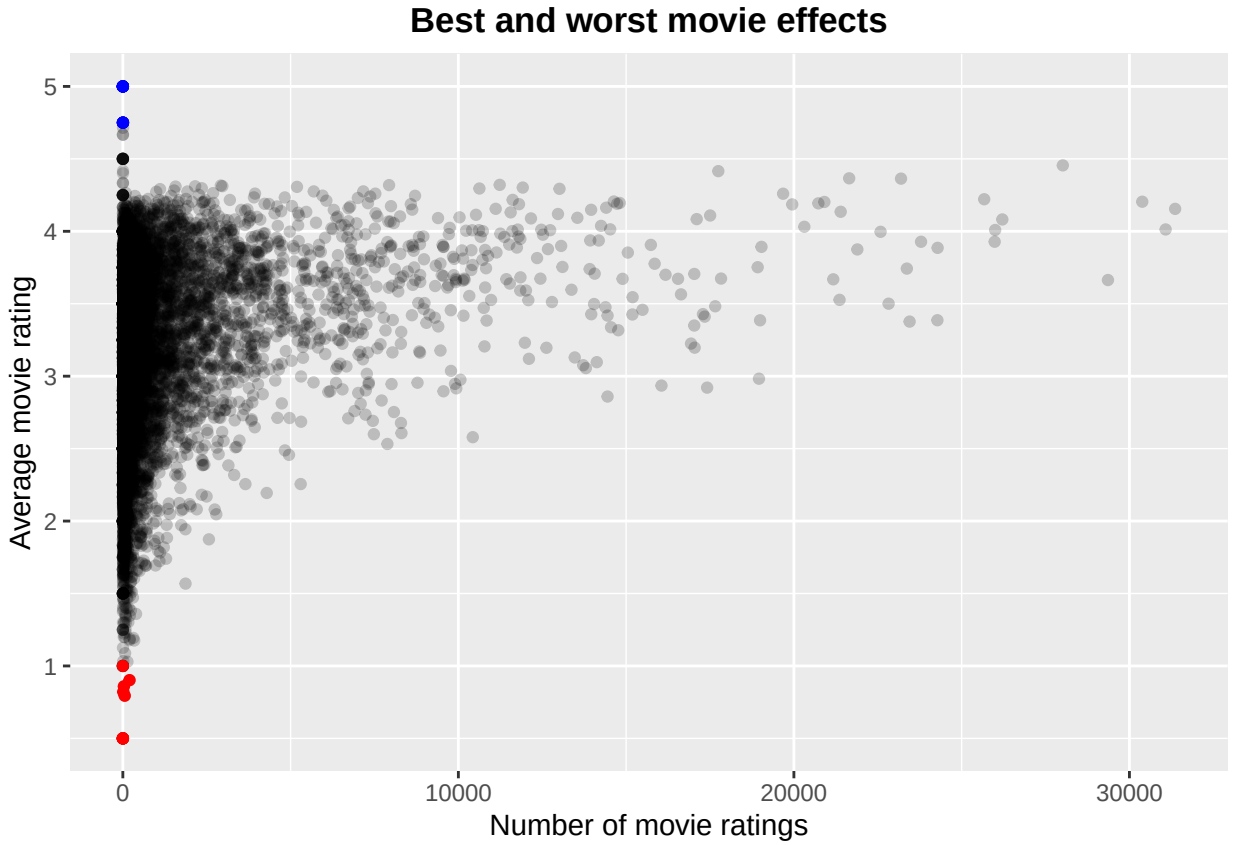
```
#Get the movies with the bottom 10 average scores
true_bottom_10 <- avg_movie_ratings %>%
  arrange(avg_rating) %>%
  slice(1:10)
true_bottom_10
```

```
## # A tibble: 10 x 3
```

```
##    movieId avg_rating num_ratings
##      <dbl>      <dbl>       <int>
##  1    5805      0.5             2
##  2    8394      0.5             1
##  3   61768      0.5             1
##  4   63828      0.5             1
##  5   64999      0.5             2
##  6    8859      0.795          56
##  7    7282      0.821          14
##  8   61348      0.859          32
##  9    6483      0.902         199
## 10     604      1               2
```

See also the next plot where the best movies are marked as blue and the worst as red:

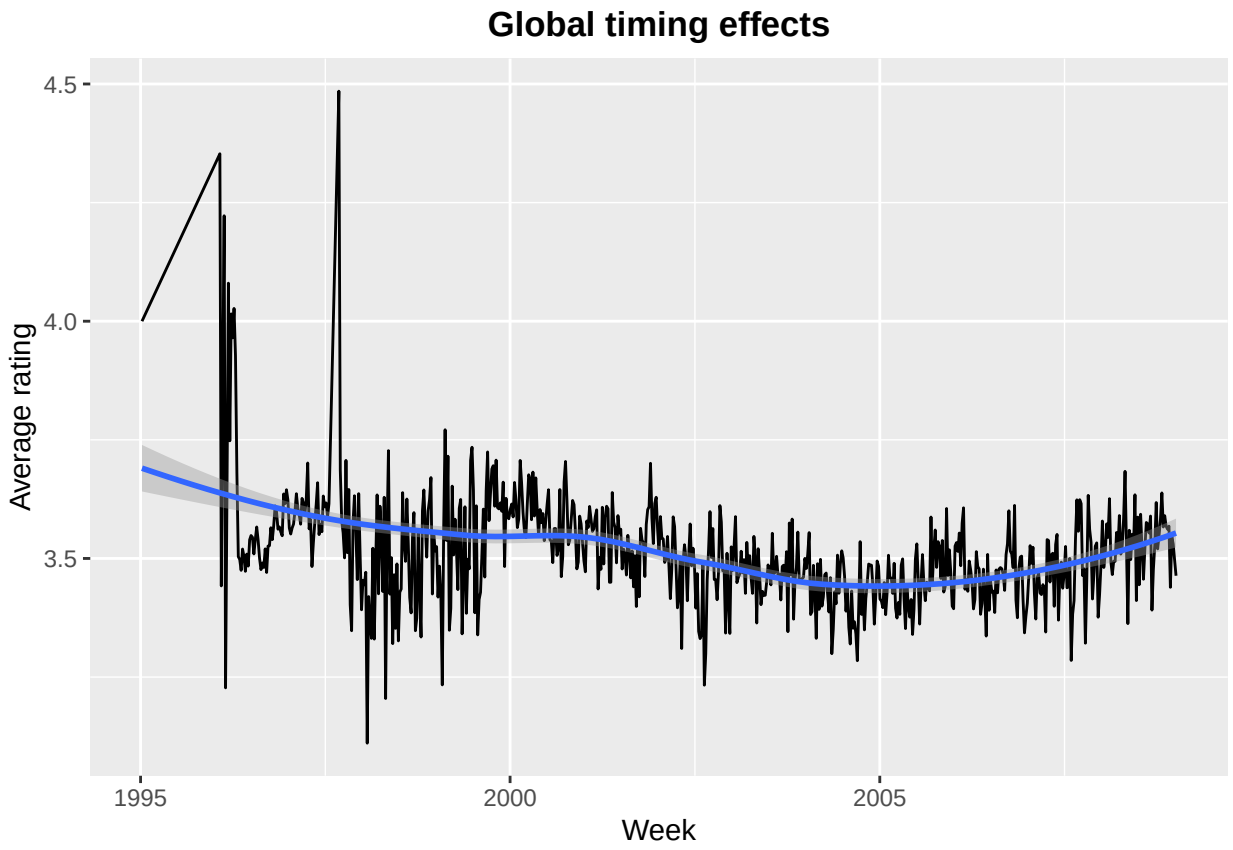
```
#Plot the movies
avg_movie_ratings %>%
  ggplot(aes(x = num_ratings, y = avg_rating)) +
  geom_point(aes(color = I("black")), alpha=0.2)+
  geom_point(data = true_top_10, aes(num_ratings, avg_rating), color="blue") +
  geom_point(data = true_bottom_10, aes(num_ratings, avg_rating), color="red") +
  labs(x="Number of movie ratings", y="Average movie rating") +
  ggtitle("Best and worst movie effects") +
  theme(plot.title = element_text(hjust = 0.5, face="bold"))
```



(O.VI): The rating scores depend on the time when the rating is given:

```
#Convert the timestamp into a week (date) and then plot
#the mean rating for all the movies per week
```

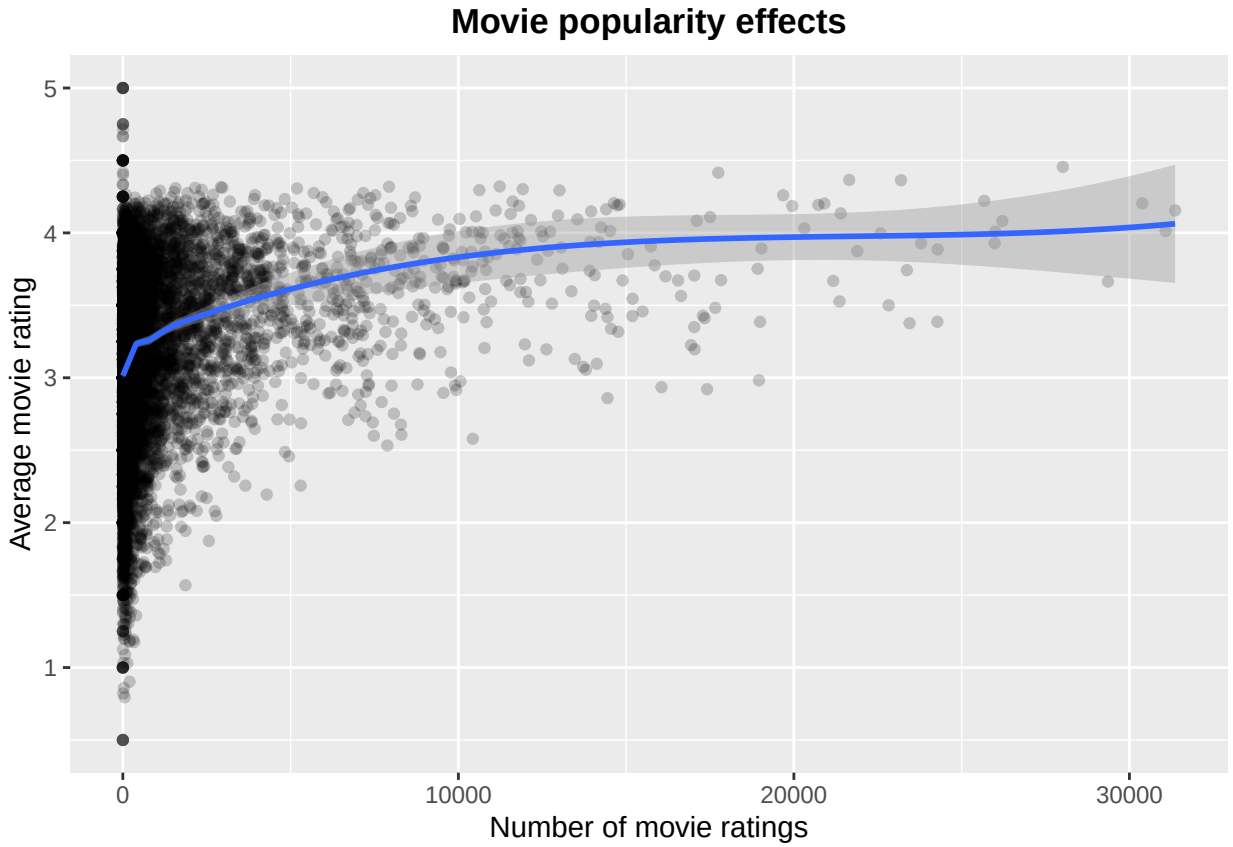
```
movielens_data$edx %>%
  mutate(date = as_datetime(timestamp), week = round_date(date, "weekId")) %>%
  group_by(weekId) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(weekId, avg_rating)) +
  geom_line() +
  geom_smooth(method = 'loess', method.args=list(degree=2)) +
  labs(x="Week", y="Average rating") +
  ggtitle("Global timing effects") +
  theme(plot.title = element_text(hjust = 0.5, face="bold"))
```



(O.VII): The ratings show dependency from the movie's popularity is rated:

As we already see from the plot above, there is more variability for the movies with fewer ratings. Let us now consider the same plot with the smoothened average movie rating trend line:

```
#Plot the movies
avg_movie_ratings %>%
  ggplot(aes(x = num_ratings, y = avg_rating)) +
  geom_point(aes(color = I("black")), alpha=0.2) +
  geom_smooth(method = 'loess', method.args=list(degree=2)) +
  labs(x="Number of movie ratings", y="Average movie rating") +
  ggtitle("Movie popularity effects") +
  theme(plot.title = element_text(hjust = 0.5, face="bold"))
```



We can clearly see that the movies that are rated more often (we call them "popular" movies) are rated higher on-average.

In addition we could consider the effects caused by the movie titles and genres. One could think of analyzing the influence of:

- Keywords and phrases in the movie titles
- Connection between sequels, prequels, and spin-offs
- Movie genre and genre combinations

Although possible and interesting, due to the lack of time, the latter will not be considered in this study.

Modeling approach

Consider the columns of the available MovieLens data:

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

Then the conditional probability distribution that is to be estimates can be formalized as:

$$Pr\left(R \mid M = m,\ U = u,\ T = t,\ G = g,\ L = l\right)$$

Here the movie-related random variables are:

- R – the rating to be given;
- U – the user id;
- M – the movie id;
- T – the ratings timestamp;

- G – the movie genres;
- L – the movie title;

So here we could use 5 predictors do provide the estimates of R.

As discussion in the end of the *"Dataset analysis"* section, we shall NOT consider the effects caused by the genres (G) and movie titles (T). This leaves us with three predictors U, M and T and the conditional probability distribution to be estimated:

$$Pr\left(R \mid M = m,\ U = u,\ T = t\right)$$

Before we proceed, let us note that the dimension reduction techniques, e.g. Principal Component Analysis (PCA), are not very useful here as the number of predictors is small, so they will not be used. Moreover, due to the large amount of data in the dataset, the standard available statistical models in R, such as the Linear Model (`lm`) or the random trees/forest models (`randomForest`), are not usable as they all fail with exhausting the available memory. This means that we are to build an efficient model for the recommendation system almost "from scratch".

Let us now consider the data effects *(O.I) – (O.VII)* outlined in the *"Dataset analysis"* section. The first *(O.I) – (O.V)* effects are the same as observed on the subset of the MovieLens dataset included into the *dslabs* package. In fact this subset was already used for a "Recommendation system" case-study in the literature[2] and that case study has suggested a statistical model taking into account the user and movie effects along with the Regularization to account *(O.V)*.

Based on the above, we shall split our modeling approach into two parts:

1. **Base model**: The movie recommendation system explained in the literature.
2. **Timing model**: We extend the *Base model* taking into account the timing effects *(O.VI)*.

Similar to the timing effects, we could also take the movie popularity effects *(O.VII)* into account. However, as the *"Results"* section will reveal, the *Timing model* will already have the RMSE score on the `validation` set below the desired 0.8649 value, c.f. section *"Project goal"*. Therefore, the *(O.VII)* effect will not be taken into account and will be left for future work.

Further, we shall first outline the *Base model*, as present in the literature. Next, we explain how we extended towards the *Timing model*, taking the timing effects into account.

Note that: We consider the *Timing model* to be the **first** innovative part of this work, as it builds upon the existing *Base model*, and will be shown to improve predictions. The **second** innovative part will be attempting using different Regularization parameters for the user and movie effects. This is a natural extension as one can suspect that movie and user effects may be required to be penalized differently.

Base model

The *Base model* estimates the probability distribution of $Pr\left(R \mid M = m,\ U = u\right)$ and can be summarized as:

$$R_{m,u} = \mu + b_m + b_u + \epsilon_{m,u}$$

where:

- $\epsilon_{m,u}$ – independent errors, sampled from the same distribution centered at 0
- μ – the average rating of all the movies
- b_m – the movie bias for movie m
- b_u – the user bias for user u

Note that, b_m and b_u could be estimated using, for instance, the linear model (using `lm()` in `R`):

[2] Chapter 33.7: *"Recommendation systems"* of the **"Introduction to Data Science"** online book by Rafael A. Irizarry

```
lm(rating ~ as.factor(movieId) + as.factor(userId))
```

However there are two reasons for not doing this:

1. Due to the size of the MovieLens data set and the number of individual movies and users, the time needed to fit even this simple model on a regular PC can be days.
2. To account for the *(O.V)* effects, we will use Regularization that has to penalize the b_m and b_u values.

The Regularization technique, with the λ modeling parameter, is used to account for the movies for which too few ratings are given and the users that gave too few ratings. Those by default may get b_m and b_u values which are either too high or too low solely due to being based on too few observations. The base model approach therefore suggests to use the estimated μ, b_m and b_u values as follows:

$$\hat{\mu} = \frac{1}{N} \sum_{m,u} r_{m,u}$$

$$\hat{b_m} = \frac{\sum_{u=1}^{N_m} (r_{m,u} - \hat{\mu})}{\lambda + N_m}$$

$$\hat{b_u} = \frac{\sum_{m=1}^{N_u} \left(r_{m,u} - \hat{b_m} - \hat{\mu} \right)}{\lambda + N_u}$$

where:

- $r_{m,u}$ – is the rating of the movie m by the user u
- N – is the total number of movie ratings
- N_m – is the number of ratings given to the movie m
- N_u – is the number of ratings given by the user u

The optimal value of λ is to be selected during the training phase of the model. The `RMSE` optimizing value of λ_{opt} is to be used when computing the `RMSE` scores but the value of λ_{opt} can not be based on the `validation` set. Thus, we apply cross validation technique by splitting the `edx` set into the `training` and `testing` sets respectively taking 80% and 20% of the `edx` set. To avoid testing failures, the care is taken of not having users and movies in the `testing` sets that are not present in the `training` set.

The actual *Base model* movie rating estimate for movie m and user u will then be:

$$r_{m,u} = \hat{\mu} + \hat{b_m} + \hat{b_u}$$

Timing model

To account for the timing effects, we need to estimate the probability distribution of

$$Pr\left(R \mid M = m,\ U = u,\ T = t\right)$$

which can be done using the next model:

$$R_{m,u} = \mu + b_m + b_u + f\left(t_{m,u}\right) + \epsilon_{m,u}$$

or equivalently

$$R_{m,u} = \mu + b_m + b_u + b_t + \epsilon_{m,u}$$

where:

- $t_{m,u}$ – is the time at which the movie m is rated by the user u
  - We shall stratify our time per week, as was done to show the *(O.V)* effect
- $f\left(.\right)$ – a smooth function of its argument approximating the average movie ratings per week
  - We shall use the Local Polynomial Regression Fitting ($LOESS$) with the degree 2 for this

- b_t – is the correction of the mean μ by the mean rating deviation from μ in week $t = t_{m,u}$

As with the *Basic model* the b_m, b_u, and b_t coefficients are to be estimated from the data.

Let us first explain how the b_t values are computed:

```
#Compute the smooth Local Regression (LOESS)
#fit for the average movie rating per week
amrpw_fit <- train_set %>%
  group_by(weekId) %>%
  summarise(avg_rating = mean(rating)) %>%
  mutate(weekId = as.numeric(weekId)) %>%
  loess(avg_rating ~ weekId, degree = 2, data = .)

#Compute the movie rating timing effects per week:
b_ts <- train_set %>%
  mutate(amrpw = predict(amrpw_fit, as.numeric(weekId))) %>%
  group_by(weekId) %>%
  summarize(b_t = min(amrpw) - mu_hat)
```

I.e. the smooth function $f(.)$ is estimated as the polynomial surface based on the average movie rating per week, using the LOESS algorithm. The b_t values are then computed as the "smoothened average movie rating in week t", i.e. $f(t)$, minus the "overall average movie rating", given by $\hat{\mu}$.

To conclude the movie rating estimate of the *Timing model* for movie m and user u in week t is:

$$r_{m,u} = \hat{\mu} + \hat{b_t} + \hat{b_m} + \hat{b_u}$$

with the coefficients computed as follows:

$$\hat{\mu} = \frac{1}{N} \sum_{m,u} r_{m,u}$$

$$\hat{b_t} = f(b_t) - \hat{\mu}$$

$$\hat{b_m} = \frac{\sum_{u=1}^{N_m} \left(r_{m,u} - \hat{b_t} - \hat{\mu} \right)}{\lambda + N_m}$$

$$\hat{b_u} = \frac{\sum_{m=1}^{N_u} \left(r_{m,u} - \hat{b_t} - \hat{b_m} - \hat{\mu} \right)}{\lambda + N_u}$$

Please note that, since we do not apply Regularization to the $\hat{b_t}$ parameter, the clarification for the $\hat{b_m}$ and $\hat{b_u}$ formulae above is straightforward. We can obtain these by considering the *Basic model* and assume that instead of the overall mean rating $\hat{\mu}$ we use the weekly mean rating $\hat{\mu} + \hat{b_t}$. Then the $\hat{b_m}$ and $\hat{b_u}$ then we will get the corresponding formulae of the *Timing model*.

Individual lambdas tuning

Note that, for the Regularization with individual lambdas for the movie and user effects, the $\hat{b_m}$ and $\hat{b_u}$ formulae used in the *Basic* and *Timing* models will be accordingly altered as follows:

$$\hat{b_m} = \frac{\sum_{u=1}^{N_m} (r_{m,u} - \hat{\mu})}{\lambda^m + N_m}$$

$$\hat{b_u} = \frac{\sum_{m=1}^{N_u} \left(r_{m,u} - \hat{b_m} - \hat{\mu} \right)}{\lambda^u + N_u}$$

and

$$\hat{b_m} = \frac{\sum_{u=1}^{N_m} \left(r_{m,u} - \hat{b_t} - \hat{\mu} \right)}{\lambda^m + N_m}$$

$$\hat{b_u} = \frac{\sum_{m=1}^{N_u} \left(r_{m,u} - \hat{b_t} - \hat{b_m} - \hat{\mu} \right)}{\lambda^u + N_u}$$

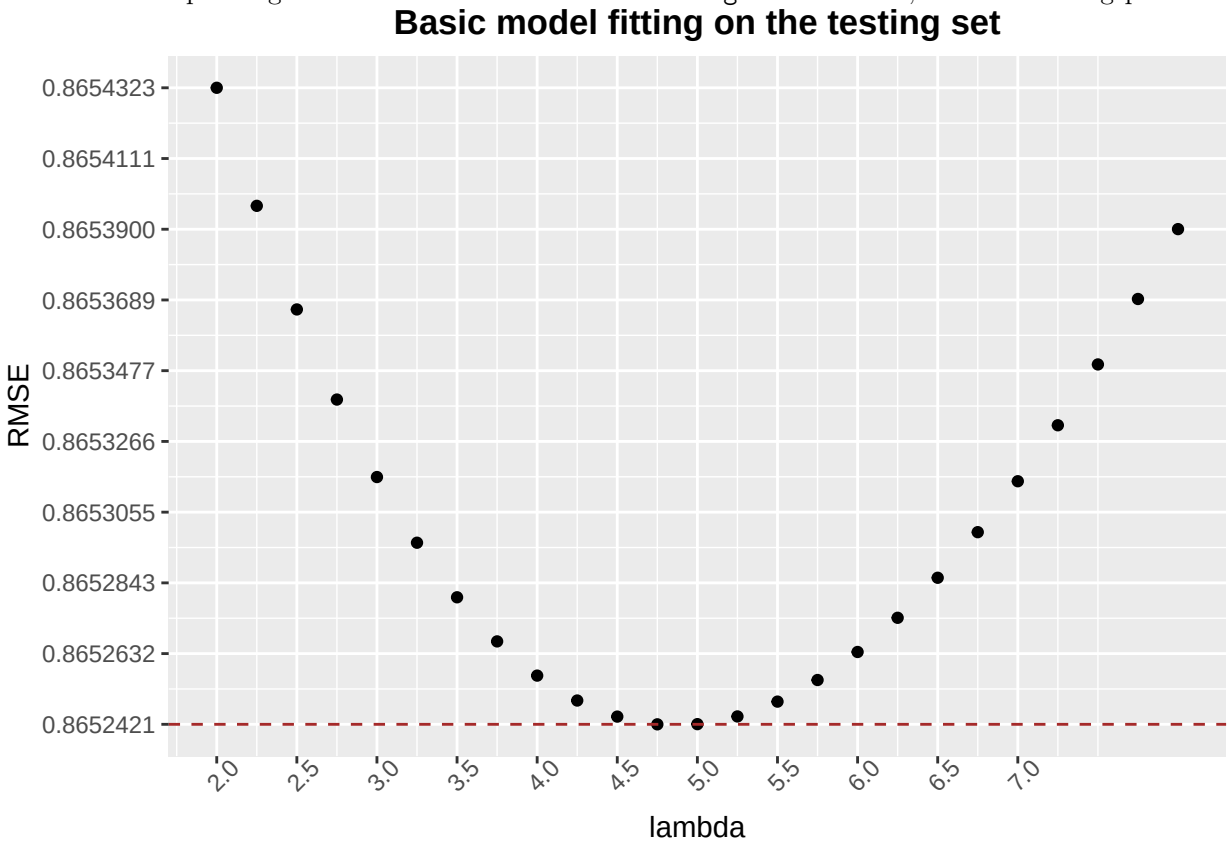
The parameter tuning will be then done on a two dimensional grid to find a pair of RMSE optimal lambda values λ^m_{opt} and λ^u_{opt}.

Results

Below we present our model tuning and validation results. The former indicate the found optimal Regularization parameter λ values. The latter provide the trained model `RMSE` scores on the `validation` set. We start with presenting the *Basic model* results and then continue with those of the *Timing model*. The results for the model extension when tuning with individual lambdas are provided in between. In the end, we also summarize the obtained results in the *"Results summary"* subsection.

Basic model

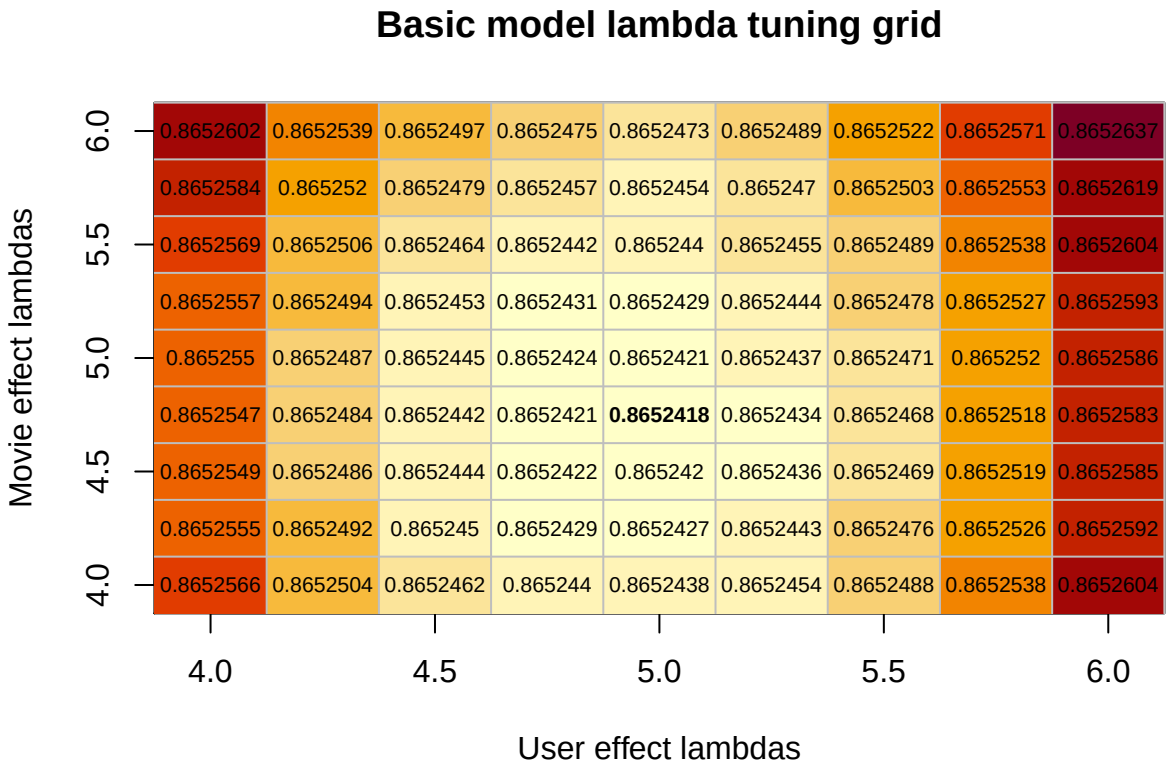
Training the *Basic model* on the range of λ in [2, 8] has revealed that the optimal value is $\lambda_{opt} = 4.75$, with the corresponding minimum `RMSE` score on the `testing` set 0.8652421, see the training plot below:



The `RMSE` score of the *Basic model* on the `validation` set is 0.8648201.

Basic model with individual lambdas

Training the *Basic model* on the range of $(\lambda^m,\ \lambda^u)$ in [4, 6]x[4, 6] has revealed that the optimal values are $\lambda^m_{opt} = 4.75$ and $\lambda^u_{opt} = 5$, with the corresponding minimum `RMSE` score on the `testing` set 0.8652418.

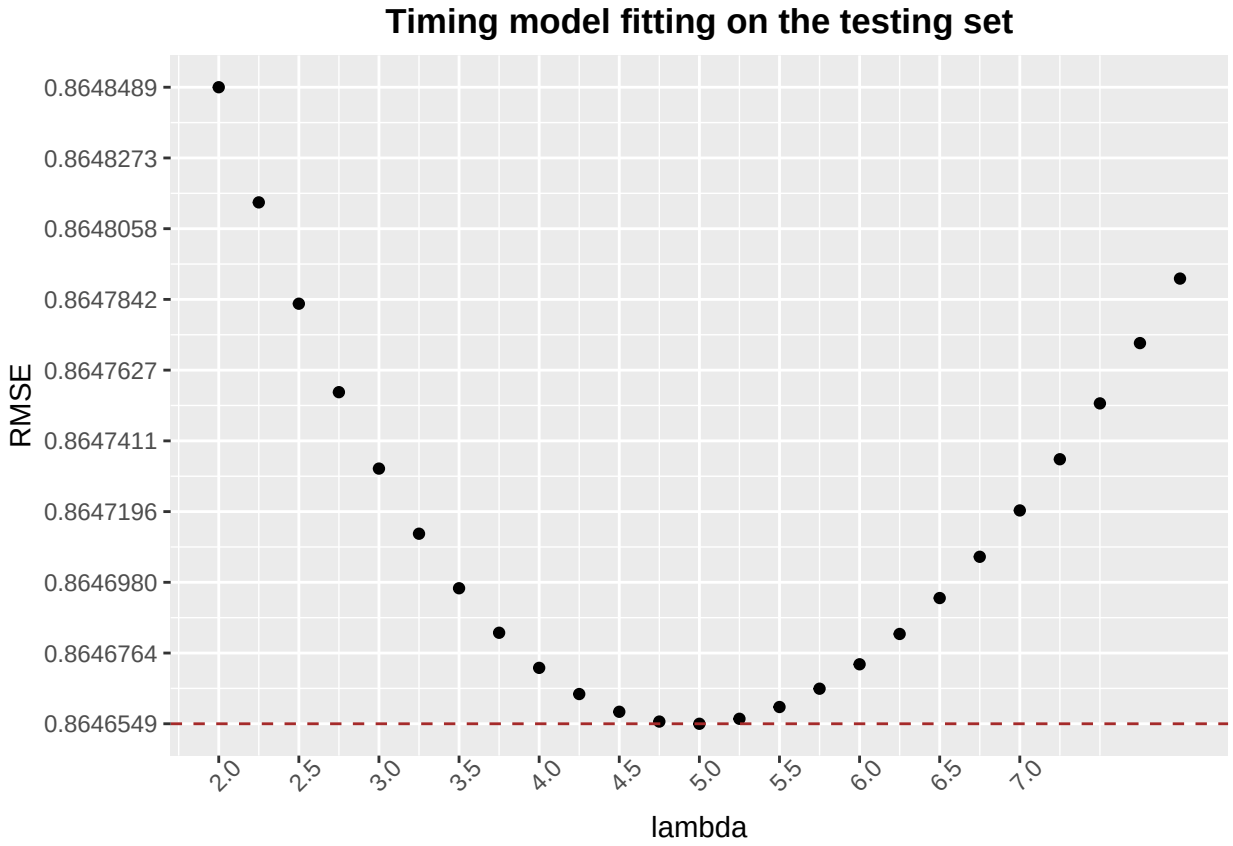


As one can see the found optimal lambda values are not equal to each other and the `RMSE` score on the `testing` set (0.8652418) is lower than that for the original *Basic model* (0.8652421).

The `RMSE` score on the `validation` set turns out to be 0.8648183, that is also smaller than the original `RMSE` of the *Basic model* (0.8648201).

Timing model

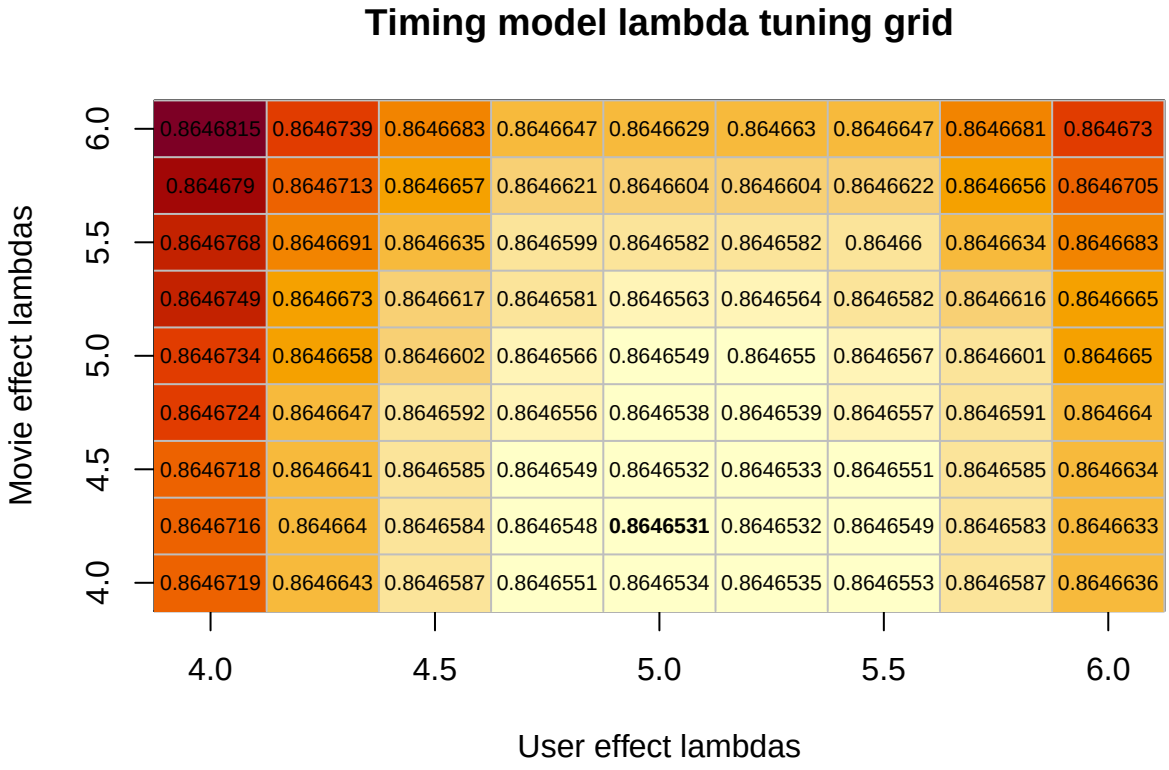
Training the *Timing model* on the range of λ in [2, 8] has revealed that the optimal value is $\lambda_{opt} = 5$, with the corresponding minimum `RMSE` score on the `testing` set 0.8646549, see the training plot below:



The `RMSE` score of the *Timing model* on the `validation` set is 0.8642035.

Timing model with individual lambdas

Training the *Timing model* on the range of (λ^m, λ^u) in [4, 6]x[4, 6] has revealed that the optimal values are $\lambda^m_{opt} = 4.25$ and $\lambda^u_{opt} = 5$, with the corresponding minimum `RMSE` score on the `testing` set 0.8646531.



As one can see the found optimal lambda values are not equal to each other and the `RMSE` score on the `testing` set (0.8646531) is lower than that for the original *Timing model* (0.8646549).

The `RMSE` score on the `validation` set turns out to be 0.8642041, that is somewhat higher than the original `RMSE` of the *Timing model* (0.8642035).

Results summary

Conclusions

Future work

- Individual lambda tuning for regularized predictors
- Account for popularity effects
- Account for genres effects
- Account for title effects (sequels, spin-offs, key words)

??? * Check for distribution of the training and validation sets * Random trees and random forests * Some standard models * Cross validation on the training set * Model ensembles