# HarvardX PH125.9x Data Science: Capstone - MovieLens Project

Edgar Pampols

6/23/2021

## Contents

# 1 Summary

This report is written within the frame of the "HarvardX - PH125.9x Data Science - Capstone" course. A dataset and some initial code to prepare is provided with the course materials, basically generating one dataset for model evaluation (named "edx") and one separate partition (named "validation", or final hold-out test set), the last one must be only used to compute the result of a final chosen model, and not for training, tuning or choosing models.

The aim of the project is to build on top of the course materials to create a model that predicts user's ratings for movies with the maximum possible precision. The project sets a target of RMSE $< 0.86490$.

The sequence of this project will consist on i) exploring different models, ii) evaluate them using the "edx" partition uniquely, iii) then decide for a final model and evaluate its performance on the "validation" set. To compare the different performance of the models, as well as computing the final result, a Residual Mean Square Error (RMSE) function will be used.

After some initial exploration of the data properties, and the construction of basic models based on the course material, further additional effects and regularization techniques have been incorporated.

The final selected model, in this report (named "Model 5") incorporates a combination of effects (movie, user, genre, age, hour and weekday effects), it was trained on the "edx" data partition alone and regularization technique was applied using cross-validation to tune parameter $\lambda$:

$$\hat{Y} = \mu + b_{i,\lambda} + b_{u,\lambda} + +b_{g,\lambda} + b_{a,\lambda} + b_{h,\lambda} + b_{d,\lambda} \quad with\ \lambda = 4.75$$

The final RMSE obtained in this project when running the final model on the validation resulted in 0.8644685 which satisfactorily achieves the project target of RMSE $< 0.86490$. This result has been achieved by applying techniques from the course.

As an additional analysis beyond the scope of the course, a complementary analysis using "recosystem" (a Matrix Factorization package) has been performed and demonstrated an even lower RMSE of 0.7822679 but this will require very long computing resources.

# 2 Data Loading, Preparation and Exploratory analysis

## 2.1 Initial data loading as per the project instructions

All the raw data for this project can be found in the following URL:

http://files.grouplens.org/datasets/movielens/ml-10m.zip

After executing the provided code, the two initial datasets "edx" and "validation" are obtained.

Here's an example of the data structure of the "edx" set:

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046          Bird of Prey (1996)
## 2:      1     185      5 838983525              Bad Moon (1996)
## 3:      1     292      5 838983421 Arsenic and Old Lace (1944)
##                       genres
## 1:                    Action
## 2: Action|Adventure|Horror
## 3: Comedy|Mystery|Thriller
```

Basically each record represent a movie rating associated to a user, with a timestamp and a movie identification number. The movie title together with movie year release are also included in the last field, as well as the movie genre or combination of genres.

The rating given from a given user to a given movie in a given time, ranges from 0.5 to 5.0.

The "validation" set will have similar structure, but will be kept unaltered until final results check.

## 2.2 Further data preparation

In order to easily evaluate RMSE for the different models, comparing predictions and test data, an RMSE function is defined as follows:

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

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

Before getting into the modeling part, a few mutations and reformatting of the data have been done to ease the analysis.

Converting the timestamp integer into year, weekday and hour of the rating:

```
##    timestamp rating_year weekday hour
## 1: 838985046        1996  Friday   14
## 2: 838983525        1996  Friday   13
## 3: 838983421        1996  Friday   13
```

Extracting the movie year of release and calculating the age of the movie at the time of rating:

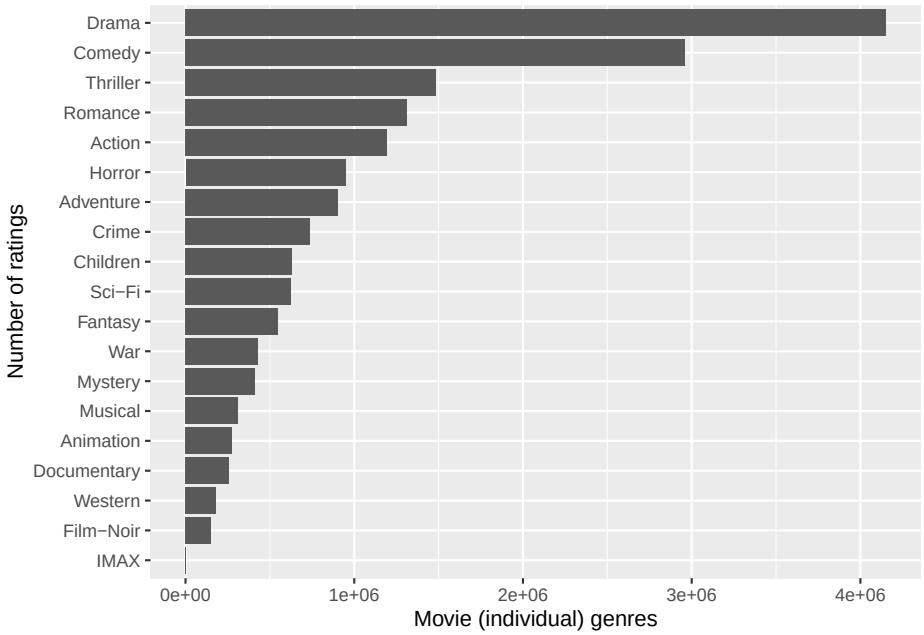
```
##                          title rating_year movie_year movie_age
## 1:         Bird of Prey (1996)        1996       1996         0
## 2:             Bad Moon (1996)        1996       1996         0
## 3: Arsenic and Old Lace (1944)        1996       1944        52
```

Additionally, a version of "edx" with the genres column split into different genres has been created, when genres are combined, to have a better understanding of genres individual contribution.

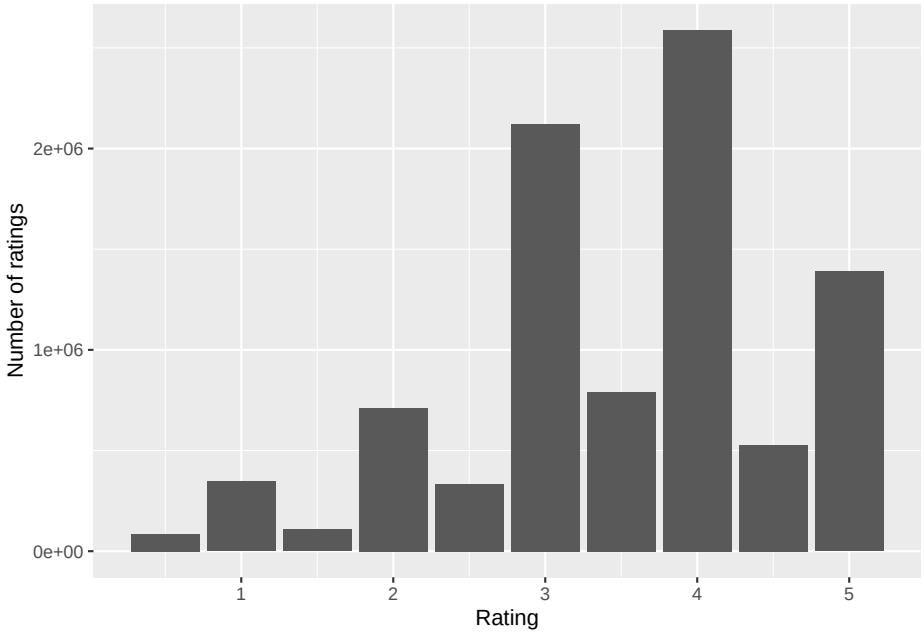
## 2.3 Selected exploratory analysis

The working dataset "edx" contains a total of ~9 million rows each corresponding to one rating. The data contains ~10,000 unique movies, ~70,000 unique users, ~700 unique combinations of genres, formed by 20 individual genre categories. The mean of a movie rating of out of 5 is ~3.5, with an average of ~130 ratings per user, and ~850 ratings per movie.

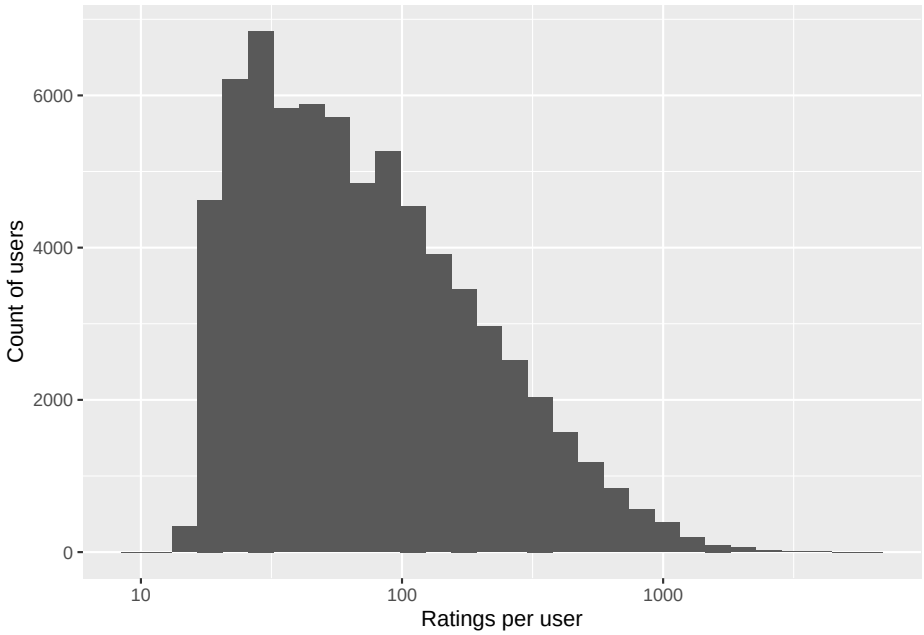
The most predominant individual genres among ratings are Drama and Comedy.



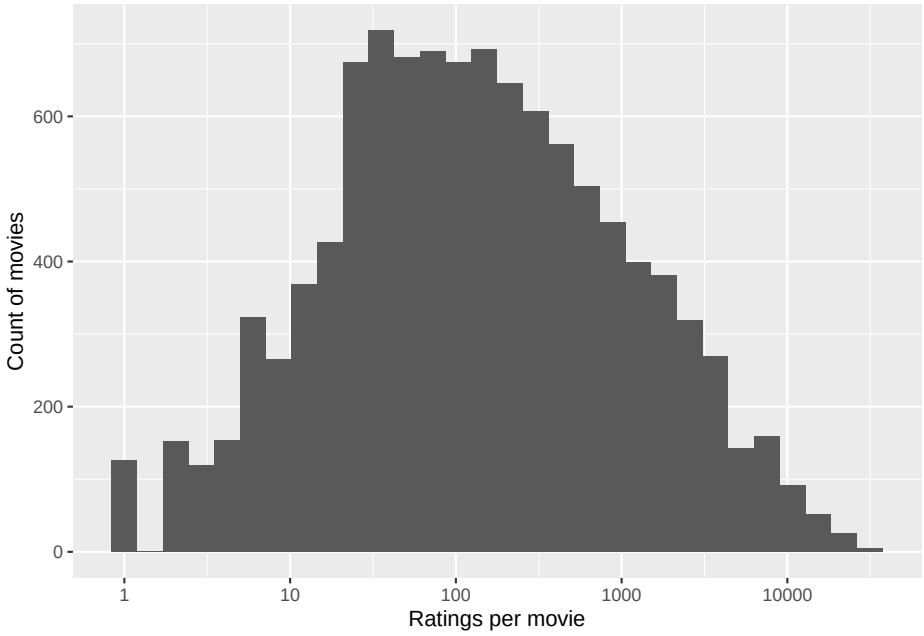

The full rounded figures (as in "x.0") seem to be preferably assigned by users to movies as opposed to give half ratings (as in "x.5").




While an average of ~130 ratings per user is observed, most of them submitted much less ratings and a few contributed with up to thousand ratings.




Similarly, certain movies have accumulated much more ratings than others.

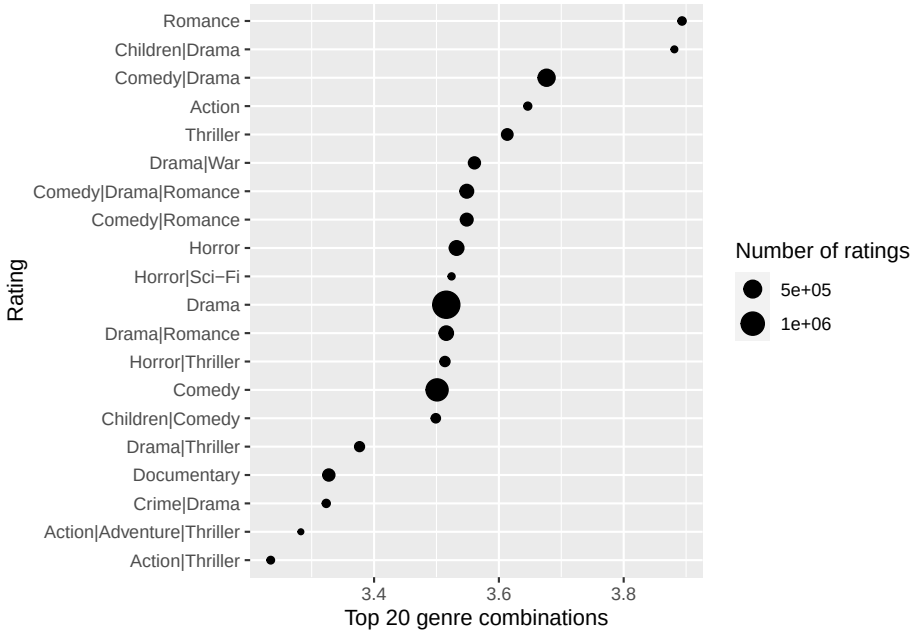



Both the amount of ratings and the mix of genres evolves and changes over the years.


It is expected that different movies have different ratings, and this will be considered in the modelling phase.

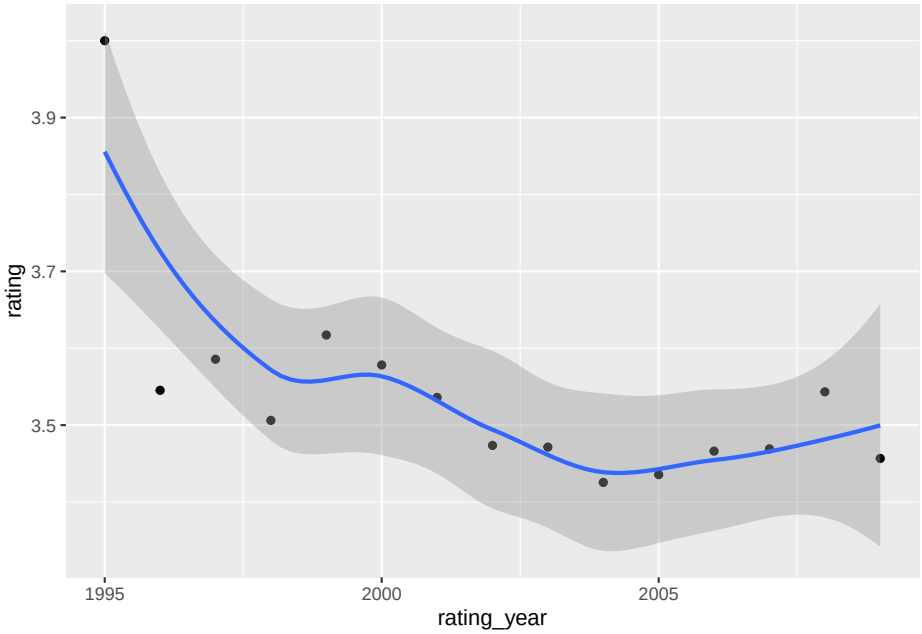
However, the following plot shows differences in ratings between different genres combinations (the top 20 genre combinations by number of ratings are shown here), while the size of the bubbles reflects the number of ratings under each genre combination.

By looking at Drama, Comedy and Comedy|Drama combination, it is observed that preserving the combined genres and treat them as an individual genres seems appropriate for the analysis. From this analysis it seems plausible to explore a "genre effect" approach while modeling in the next chapters.

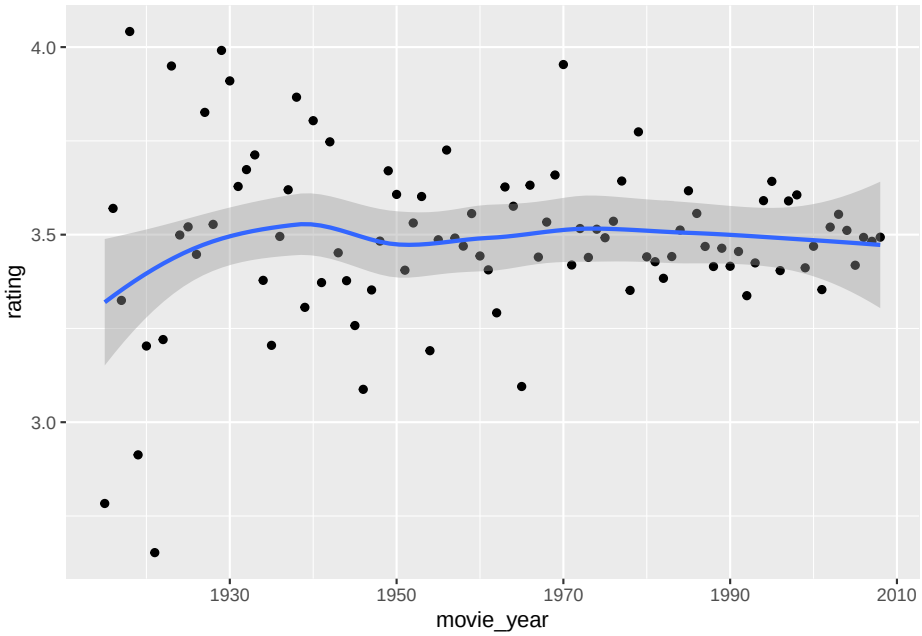


The following exploratory views, illustrate deviations in rating by different variables other than the movies themselves.

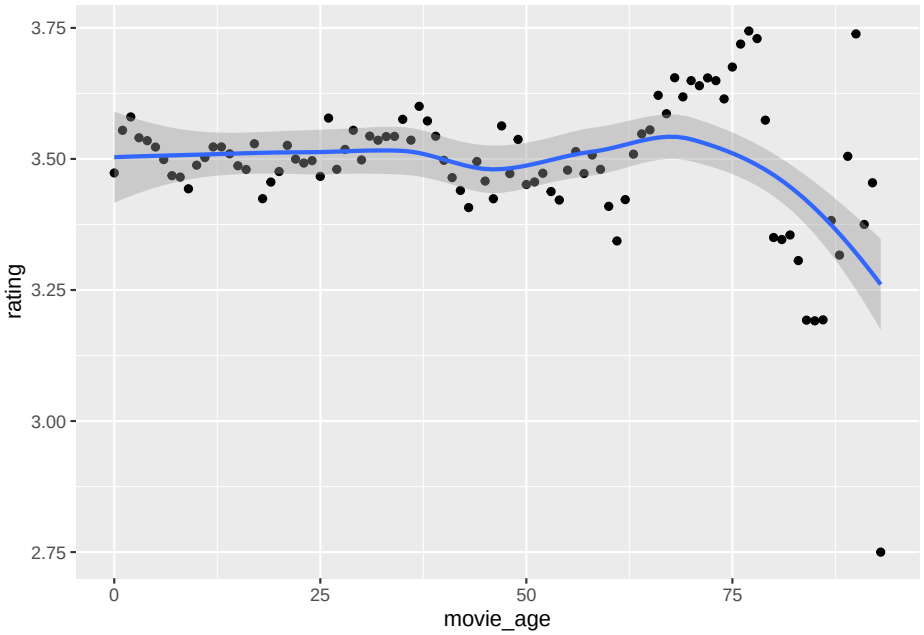
Here's a view of how the average rating has evolved across the year the rating was created.



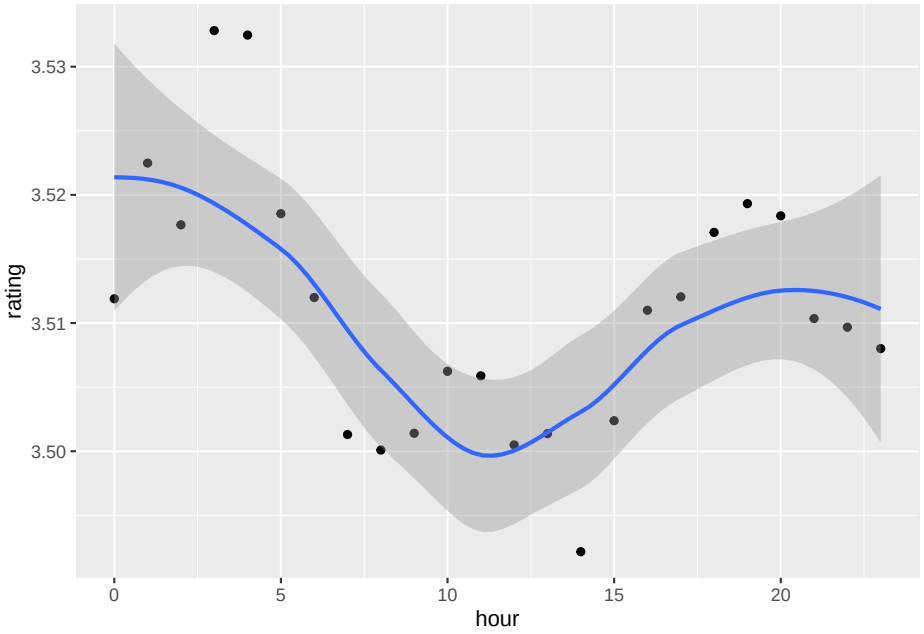

Similarly, the following of average rating by the year the movie was released suggests that older movies get higher ratings.



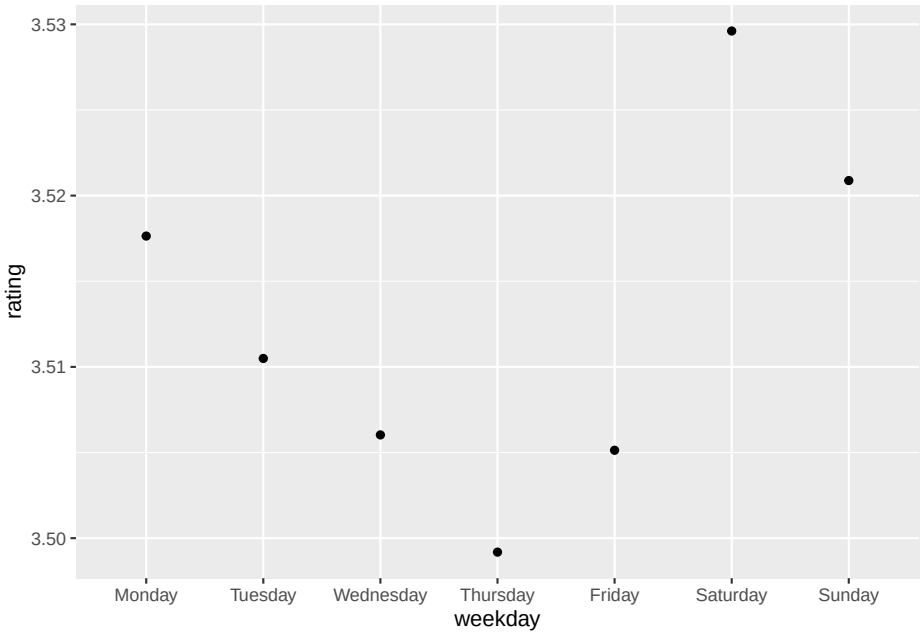

Combining the previous views to compute the age of the movie by the year the ratings was generated shows how old movies (for example more than 50 years old) get higher ratings, except for extremely old ones (more than 75 years old).




There seems to be a relatively smooth variation in ratings depending on what time of the day the rating was generated, with evening and night times leading to higher ratings.




Similarly, ratings generated on weekend days result into higher evaluations.




The variations in rating are not very relevant, but all these effects combined can help make the model more precise. One practical point to consider will be the frequency in refreshing the model (i.e. daily, weekly or real-time) depending on which the time of day or the day of the week the rating is generated can play a role or not.

# 3 First set of basic models

Before starting any model development and comparison, it is required to repartition the "edx" data set into a training and a testing subsets, the following code is used for that purpose:

```
# Test set will be 10% of edx and train set will be the remining
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)

train_temp <- edx[-test_index,]
test_temp <- edx[test_index,]

# Making sure userId and movieId in test set are also in training set
test_edx <- test_temp %>% semi_join(train_temp, by = 'movieId') %>%
  semi_join(train_temp, by = 'userId')

removed <- anti_join(test_temp, test_edx)
train_edx <- rbind(train_temp, removed)
```

This is a crucial step in model development and a key requirement for the project submission.

## 3.1 Development of basic models

A simple model based on assigning the rating average to all the ratings is built and evaluated as a first reference.

$$\hat{Y} = \mu$$

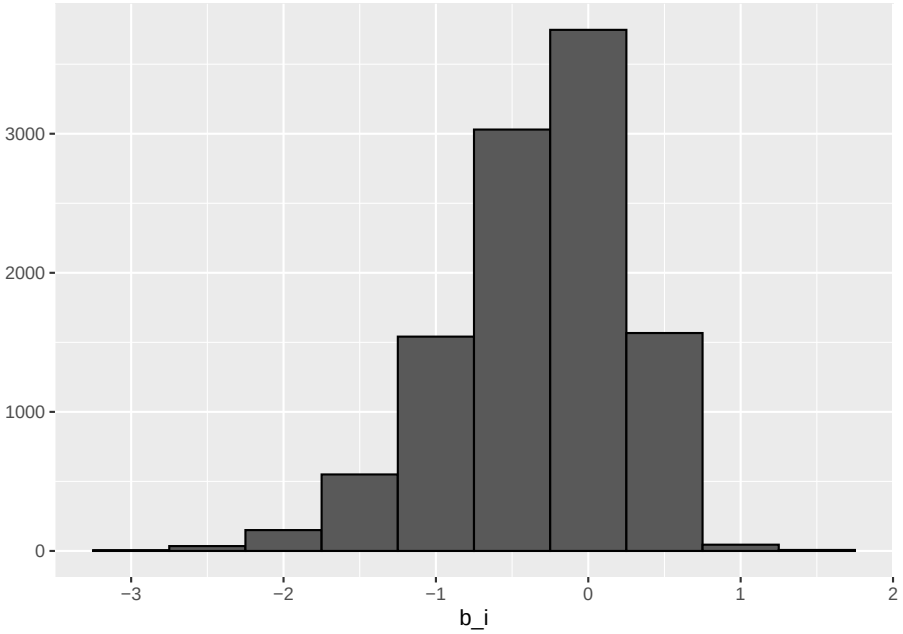
Below is the RMSE of "Model 0" where the ratings prediction is missed by more than one star.

| method | RMSE |
|---|---|
| Model 0: Just the average | 1.061135 |

The next are models to evaluate the stand-alone effect of different variables. The predicted rating is improved by adding or subtracting a particular variable effect from $\mu$. Here's an example of "Model 1a: Movie Effect alone" with its simple code and an illustration of the effect of $b_i$ over $\mu$:

$$\hat{Y} = \mu + b_i$$

```
#MODEL 1a --> factoring for the movie effect alone

movie_avgs <- train_edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```




```
predicted_ratings <- mu + test_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
```

The same approach with identical code adapted to each variable, will be used to evaluate other variable effects such as gender, hour, user, age of the movie or weekday. For which the following RMSE results are obtained:

| method | RMSE |
|---|---|
| Model 0: Just the average | 1.0611350 |
| Model 1a: Movie Effect alone Model | 0.9441568 |
| Model 1b: User Effect alone Model | 0.9795916 |

| method | RMSE |
|---|---|
| Model 1c: Hour Effect alone Model | 1.0611016 |
| Model 1d: Age Effect alone Model | 1.0601794 |
| Model 1e: Genres Effect alone Model | 1.0411427 |

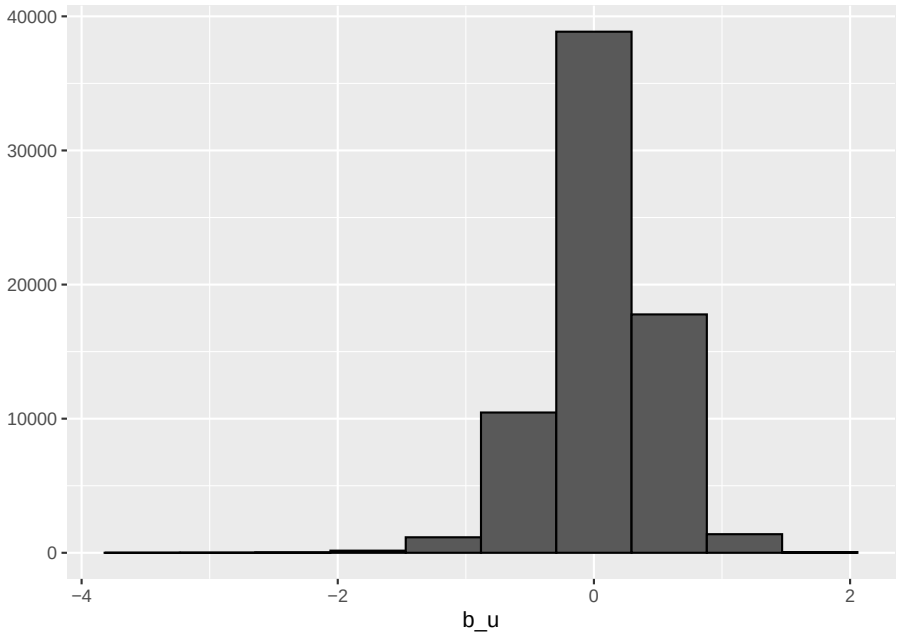
Movie and user effects, followed by genre, are the variables that improve better the RMSE results. Age, hour and other tested effects drive some results but comparatively marginal.

## 3.2 Combining multiple effects

The following tests will involve combining multiple effects of the biases from different variables. The predicted rating is improved by further adding or subtracting additional effects. Here's an example of "Model 2a: User Effect on top of Movie Effect" with its code and an illustration of the effect of $b_i$ and $b_u$ over $\mu$:

$$\hat{Y} = \mu + b_i + b_u$$

```
user_avgs <- train_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```




```
predicted_ratings <- test_edx %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

Subsequently, additional effects can be modelized, in this case the order chosen has been based on the order of RMSE performance of each stand-alone model in previous section. As illustrated later, "Model 2a" performed better than "Model 2b", where the only difference was the order of the effects calculation.

Here's an example of "Model 2c: Movie/User/Genre/Age/Hour/Day Effects" combining multiple effects:

$$\hat{Y} = \mu + b_i + b_u + + b_g + b_a + b_h + b_d$$

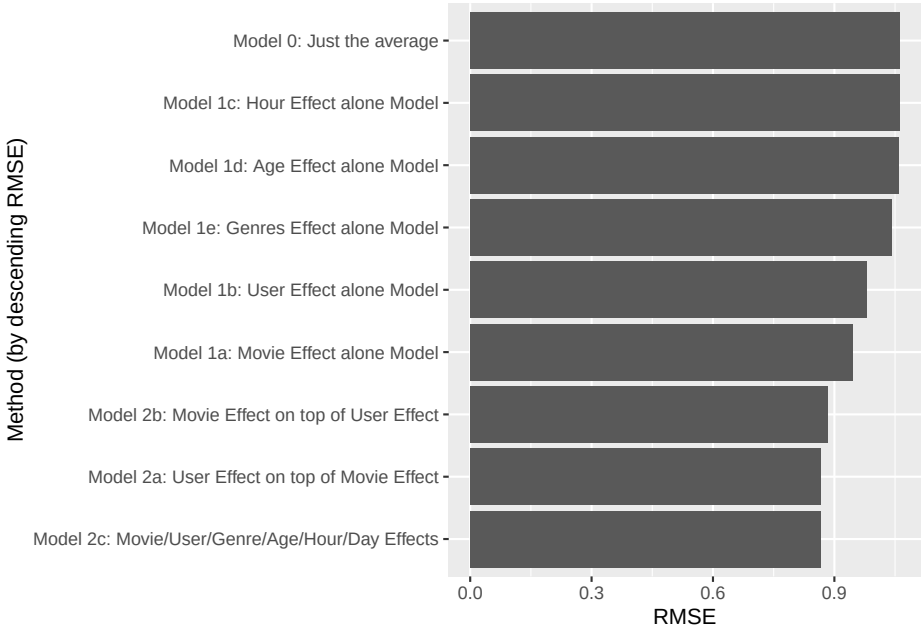
```
#MODEL 2c --> factoring for the year, hour and day effect on top of movie and user effects

#the effects are added in order of performance of the standalone models

movie_avgs <- train_edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

user_avgs <- train_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

genre_avgs <- train_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

age_avgs <- train_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(movie_age) %>%
  summarize(b_a = mean(rating - mu - b_i - b_u - b_g))

hour_avgs <- train_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(age_avgs, by='movie_age') %>%
  group_by(hour) %>%
  summarize(b_h = mean(rating - mu - b_i - b_u - b_g - b_a))

weekday_avgs <- train_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(age_avgs, by='movie_age') %>%
  left_join(hour_avgs, by='hour') %>%
  group_by(weekday) %>%
  summarize(b_w = mean(rating - mu - b_i - b_u - b_g - b_a - b_h))

predicted_ratings <- test_edx %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(age_avgs, by='movie_age') %>%
  left_join(hour_avgs, by='hour') %>%
  left_join(weekday_avgs, by='weekday') %>%
```

```
  mutate(pred = mu + b_i + b_u + b_g + b_a + b_h + b_w) %>%
  .$pred
```

Below are the RMSE results of the models developed so far, it is visible how combining effects improves rating prediction.

| method | RMSE |
|---|---|
| Model 0: Just the average | 1.0611350 |
| Model 1a: Movie Effect alone Model | 0.9441568 |
| Model 1b: User Effect alone Model | 0.9795916 |
| Model 1c: Hour Effect alone Model | 1.0611016 |
| Model 1d: Age Effect alone Model | 1.0601794 |
| Model 1e: Genres Effect alone Model | 1.0411427 |
| Model 2a: User Effect on top of Movie Effect | 0.8659736 |
| Model 2b: Movie Effect on top of User Effect | 0.8826201 |
| Model 2c: Movie/User/Genre/Age/Hour/Day Effects | 0.8655933 |

"Model 2c" is the best performing model among simple models, so far.




# 4 Applying regularization

While observing the largest errors from our last models (see tables below), they concentrate on movies that have very small number of ratings (i.e. 1 rating only). This observations act like noise to the predictions.

```
## # A tibble: 5 x 3
##   title                                     n error
##   <chr>                                 <int> <dbl>
## 1 Fearless (Huo Yuan Jia) (2006)            1  3.80
```

```
## 2 Guardian, The (2006)                     1  3.15
## 3 In the Good Old Summertime (1949)        1  2.94
## 4 Keeping the Faith (2000)                 1  2.65
## 5 T-Rex: Back to the Cretaceous (1998)     1  2.93
```

```
## # A tibble: 5 x 3
##   title                                                                n error
##   <chr>                                                            <int> <dbl>
## 1 Evolution (2001)                                                     1 -2.84
## 2 Good Boy! (2003)                                                     1 -3.32
## 3 House on Haunted Hill (1999)                                         1 -2.88
## 4 Once Upon a Time in China II (Wong Fei-hung Ji Yi: Naam yi dong j~    1 -3.38
## 5 State Property (2002)                                                1 -2.85
```

## 4.1 Regularization principles

The general idea behind regularization is to constrain the total variability of the effect sizes and penalize instances were number of observations is very small by adding a parameter $\lambda$ to our models. For example, to improve the previously simple "Model 1a: Movie Effect alone Model" the values of the previous $b_i$ are recalculated as $b_{i,\lambda}$ like in the expression below:

$$b_i = \frac{1}{n_i}\sum_{u=1}^{n_i}(Y_{u,i} - \mu) \implies b_{i,\lambda} = \frac{1}{\lambda + n_i}\sum_{u=1}^{n_i}(Y_{u,i} - \mu) \quad with\ n_i\ number\ of\ ratings\ for\ movie\ i$$

This approach will, when the sample size $n_i$ is very large, give a stable estimate, then the penalty $\lambda$ will effectively be ignored since $n_i + \lambda \approx n_i$. However, when the $n_i$ is small, then the estimate $\hat{b}_i(\lambda)$ is shrunken towards 0.

## 4.2 Preparation of k-fold cross-validation

As the following models will require the tuning of external parameter(s) (i.e. $\lambda$) it is required to subset the previously used training data into $k$ partitions to use cross-validation. In the next iterations $k = 5$ will be used to reduce computing time.

```
#preparation of cross-validation data partitions to tune regularization parameter

nb_folds <- 5

cv_folds <- createFolds(train_edx$rating, k = nb_folds, returnTrain = TRUE)
```

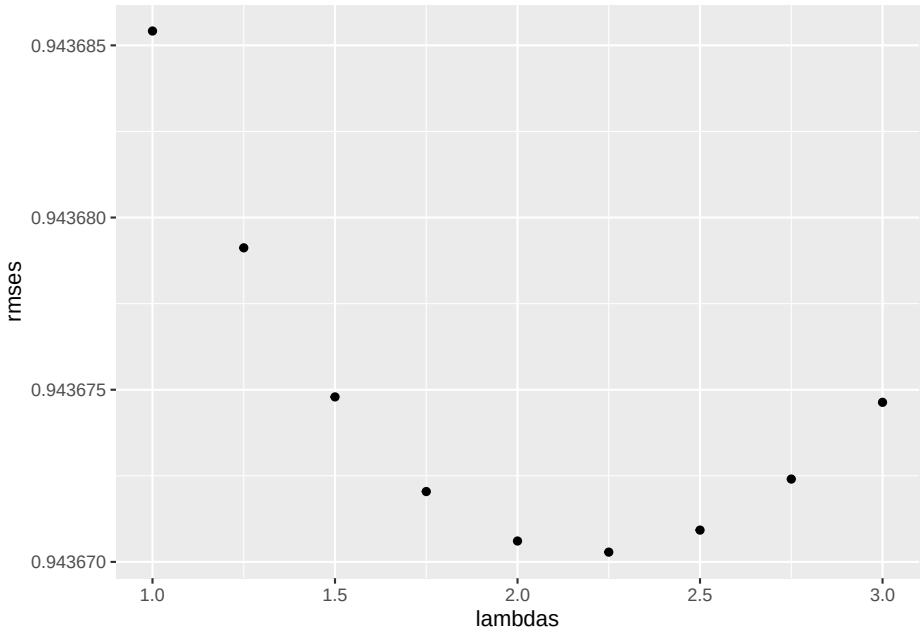
Each of the regularized models will be tuned using the "edx" test cross-validation partitions, then built on the entire "edx" training partition for more accuracy, and evaluated on the "edx" test partition for comparison (exact same test set as in the simple models in previous sections). The "validation" set will still not be used until a final model has been selected.

Below is an example of how the code for "Model 3a: Movie Effect alone regularized" is tuned by looping across the different cross-validation sets and different ranges for $\lambda$ to select the optimal parameter value.

```
#MODEL 3a --> applying regularization to movie effect

l_min <- 1
```

```
l_max <- 3
l_steps <- 0.25

rmse_matrix <- matrix(nrow=nb_folds,ncol=(l_max-l_min)/l_steps+1)
lambdas <- seq(l_min, l_max, l_steps)

for(k in 1:nb_folds) {
  train_temp <- train_edx[cv_folds[[k]],]
  test_temp <- train_edx[-cv_folds[[k]],]

  test_data <- test_temp %>% semi_join(train_temp, by = 'movieId') %>%
    semi_join(train_temp, by = 'userId')

  removed <- anti_join(test_temp, test_data)
  train_data <- rbind(train_temp, removed)

  mu <- mean(train_data$rating)

  just_the_sum <- train_data %>% group_by(movieId) %>%
    summarize(s = sum(rating - mu), n_i = n())

  rmse_matrix[k,] <- sapply(lambdas, function(l){
    predicted_ratings <- test_data %>%
      left_join(just_the_sum, by='movieId') %>%
      mutate(b_i = s/(n_i+l)) %>%
      mutate(pred = mu + b_i) %>%
      .$pred

  return(RMSE(predicted_ratings, test_data$rating))
  })
}

rmses <- colMeans(rmse_matrix)
lambda_best <- lambdas[which.min(rmses)]
```

The value for parameter $\lambda$ that optimizes the RMSE for this model is 2.25.

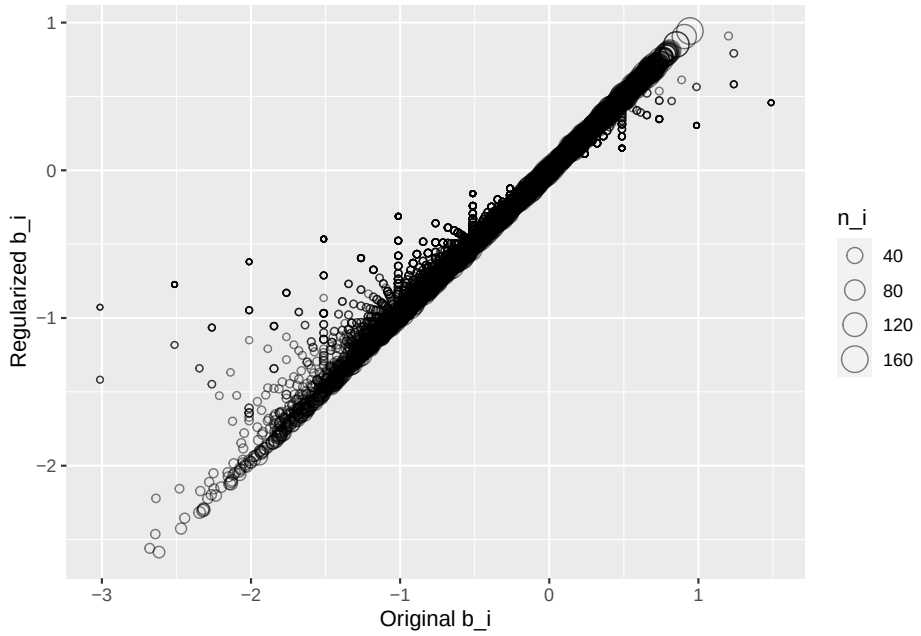



The following code shows how the optimum $\lambda$ is used on the "edx" train set to build the model and subsequently evaluate it on the "edx" test partition.

```
#build the model and test with edx test partition set

mu <- mean(train_edx$rating)

movie_avgs <- train_edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda_best), n_i = n())
```

```
predicted_ratings <- mu + test_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
```

Note how the regularization now penalizes the $b_i$ values for the movies with very low number of ratings, making them converge towards zero, as appreciated in the below plot.




| method | RMSE |
|---|---|
| Model 1a: Movie Effect alone Model | 0.9441568 |
| Model 3a: Movie Effect alone regularized | 0.9441164 |

The RMSE of "Model 3a: Movie Effect alone regularized" improves slightly compared to its non-regularized equivalent.

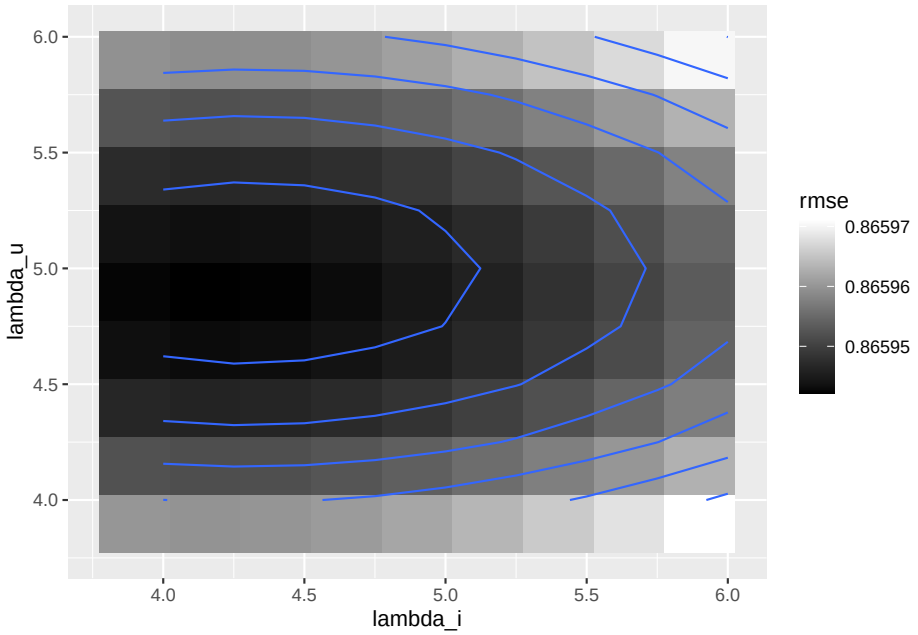
## 4.3 Building on regularized models

Based on the regularization technique illustrated above, a series of more evolved models have been tested, like adding additional effects to regularization, or testing the effect of adding separate customized $\lambda_i$ and $\lambda_u$ to see if RMSE improves compared to having the same $\lambda$ for each effect.

The principle is to calculate the additional effects as follows (an example of user effect on top of movie effect is used here):

$$b_{u,\lambda} = \frac{1}{\lambda + n_u} \sum_{u=1}^{n_u} (Y_{u,i} - \mu - b_{i,\lambda})$$

Attempting to regularize movie and user effect tuning different $\lambda$ values for movies and users, results in $\lambda_i$ and $\lambda_u$ with very close values to the optimal unique $\lambda$ value version.




These are the results of the aforementioned tests. Adding effects to the model improves the RMSE, however customizing a parameter lambda for movies and users separately did not seem to improve much the results, while increasing heavily the computation time to tune combinations of two parameters.

| method | RMSE |
|---|---|
| Model 3a: Movie Effect alone regularized | 0.9441164 |
| Model 3b: Movie then User Effect regularized | 0.8654678 |
| Model 4: Different lambdas to Movie first then to User Effects | 0.8654669 |

| method | Lambda |
|---|---|
| Model 3a: Lambda_i | 2.25 |
| Model 3b: Lambda_i_u | 4.75 |
| Model 4: Lambda_i | 4.25 |
| Model 4: Lambda_u | 5.00 |

Based on the above results, the next model attempt will accumulate a maximum number of effects, in order of stand-alone impact, and will be regularized by a single $\lambda$ parameter.

# 5 Final project model

The following, and last model, which turns to be the best performing one, leverages on the learnings of the previous tests.

## 5.1 Development and choice justification

"Model 5: Movie/User/Genre/Age/Hour/Day Effects regularized" incorporates a combination of effects (movie, user, genre, age, hour and weekday effects), and regularization via a parameter $\lambda$:

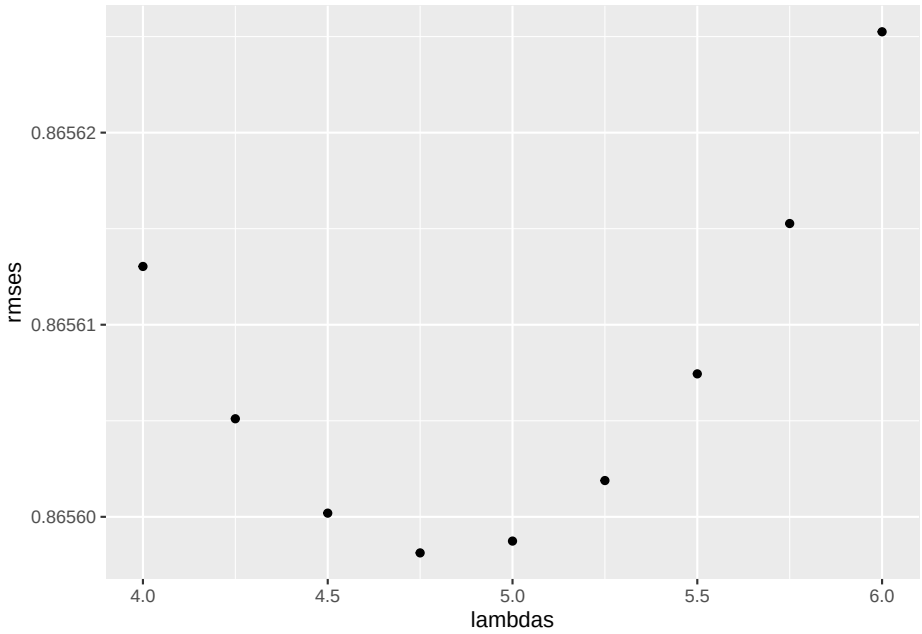
$$\hat{Y} = \mu + b_{i,\lambda} + b_{u,\lambda} + +b_{g,\lambda} + b_{a,\lambda} + b_{h,\lambda} + b_{d,\lambda}$$

Below is the code used to tune $\lambda$ for this model using cross-validation, and compare it to the previously developed models, to verify its performance in the test environment (no use of validation data yet).

```
#MODEL 5 --> applying regularization to all effects calculated before

l_min <- 4
l_max <- 6
l_steps <- 0.25

rmse_matrix <- matrix(nrow=nb_folds,ncol=(l_max-l_min)/l_steps+1)
lambdas <- seq(l_min, l_max, l_steps)

for(k in 1:nb_folds) {
  train_temp <- train_edx[cv_folds[[k]],]
  test_temp <- train_edx[-cv_folds[[k]],]

  test_data <- test_temp %>% semi_join(train_temp, by = 'movieId') %>%
    semi_join(train_temp, by = 'userId')

  removed <- anti_join(test_temp, test_data)
  train_data <- rbind(train_temp, removed)

  mu <- mean(train_data$rating)

  rmse_matrix[k,] <- sapply(lambdas, function(l){
    movie_avgs <- train_data %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+l))

    user_avgs <- train_data %>%
      left_join(movie_avgs, by='movieId') %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - mu - b_i)/(n()+l))

    genre_avgs <- train_data %>%
      left_join(movie_avgs, by='movieId') %>%
      left_join(user_avgs, by='userId') %>%
      group_by(genres) %>%
      summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l))

    age_avgs <- train_data %>%
      left_join(movie_avgs, by='movieId') %>%
      left_join(user_avgs, by='userId') %>%
      left_join(genre_avgs, by='genres') %>%
      group_by(movie_age) %>%
      summarize(b_a = sum(rating - mu - b_i - b_u - b_g)/(n()+l))

    hour_avgs <- train_data %>%
      left_join(movie_avgs, by='movieId') %>%
      left_join(user_avgs, by='userId') %>%
      left_join(genre_avgs, by='genres') %>%
      left_join(age_avgs, by='movie_age') %>%
      group_by(hour) %>%
      summarize(b_h = sum(rating - mu - b_i - b_u - b_g - b_a)/(n()+l))
```

```
    weekday_avgs <- train_data %>%
      left_join(movie_avgs, by='movieId') %>%
      left_join(user_avgs, by='userId') %>%
      left_join(genre_avgs, by='genres') %>%
      left_join(age_avgs, by='movie_age') %>%
      left_join(hour_avgs, by='hour') %>%
      group_by(weekday) %>%
      summarize(b_w = sum(rating - mu - b_i - b_u - b_g - b_a - b_h)/(n()+l))

    predicted_ratings <- test_data %>%
      left_join(user_avgs, by='userId') %>%
      left_join(movie_avgs, by='movieId') %>%
      left_join(genre_avgs, by='genres') %>%
      left_join(age_avgs, by='movie_age') %>%
      left_join(hour_avgs, by='hour') %>%
      left_join(weekday_avgs, by='weekday') %>%
      mutate(pred = mu + b_i + b_u + b_g + b_a + b_h + b_w) %>%
      .$pred

    return(RMSE(predicted_ratings, test_data$rating))
  })
}

rmses <- colMeans(rmse_matrix)

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

The optimal $\lambda$ value for "Model 5" results in 4.75 (similar to "Model 3b"), the tuning phase is completed.




With the calculated value for $\lambda$, let's use the "edx" train partition to compare this model to the previous.

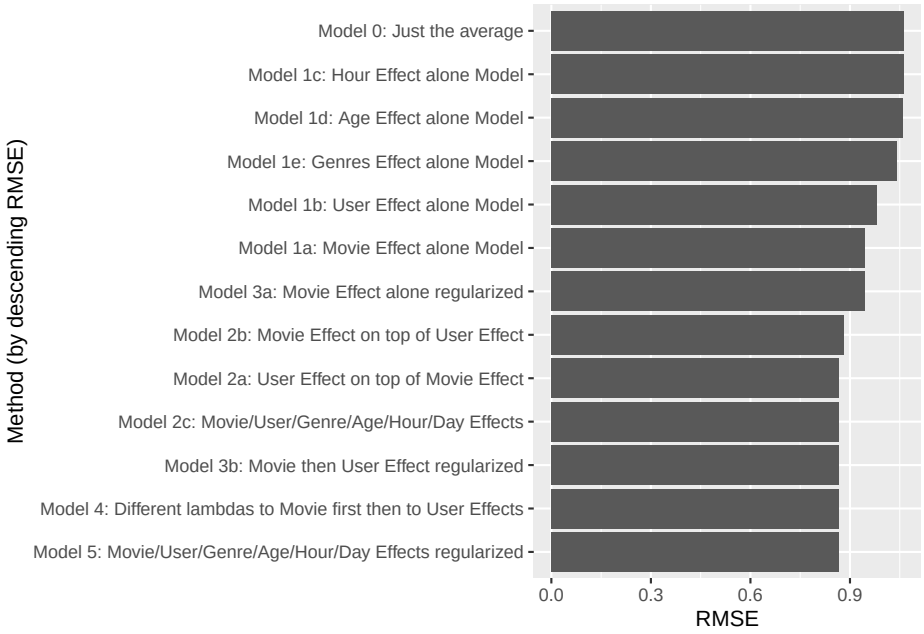
```
#build the model and test with edx test set

mu <- mean(train_edx$rating)

movie_avgs <- train_edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda_best))

user_avgs <- train_edx %>%
  left_join(movie_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda_best))

genre_avgs <- train_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+lambda_best))

age_avgs <- train_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(movie_age) %>%
  summarize(b_a = sum(rating - mu - b_i - b_u - b_g)/(n()+lambda_best))

hour_avgs <- train_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(age_avgs, by='movie_age') %>%
  group_by(hour) %>%
  summarize(b_h = sum(rating - mu - b_i - b_u - b_g - b_a)/(n()+lambda_best))

weekday_avgs <- train_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(age_avgs, by='movie_age') %>%
  left_join(hour_avgs, by='hour') %>%
  group_by(weekday) %>%
  summarize(b_w = sum(rating - mu - b_i - b_u - b_g - b_a - b_h)/(n()+lambda_best))

predicted_ratings <- test_edx %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(age_avgs, by='movie_age') %>%
  left_join(hour_avgs, by='hour') %>%
  left_join(weekday_avgs, by='weekday') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_a + b_h + b_w) %>%
  .$pred
```

```
## [1] 0.8651046
```

The RMSE obtained with "Model 5" successfully beats all other previously tested models with 0.8651064.

| method | RMSE |
|---|---|
| Model 3a: Movie Effect alone regularized | 0.9441164 |
| Model 3b: Movie then User Effect regularized | 0.8654678 |
| Model 4: Different lambdas to Movie first then to User Effects | 0.8654669 |
| Model 5: Movie/User/Genre/Age/Hour/Day Effects regularized | 0.8651046 |

| method | Lambda |
|---|---|
| Model 3a: Lambda_i | 2.25 |
| Model 3b: Lambda_i_u | 4.75 |
| Model 4: Lambda_i | 4.25 |
| Model 4: Lambda_u | 5.00 |
| Model 5: Lambda_all | 4.75 |




## 5.2 Model creation and final hold-out test

The last step consists on creating the final model with the entire "edx" set for better training, and then apply the model to the "validation" set to observe the final result on fresh unknown data.

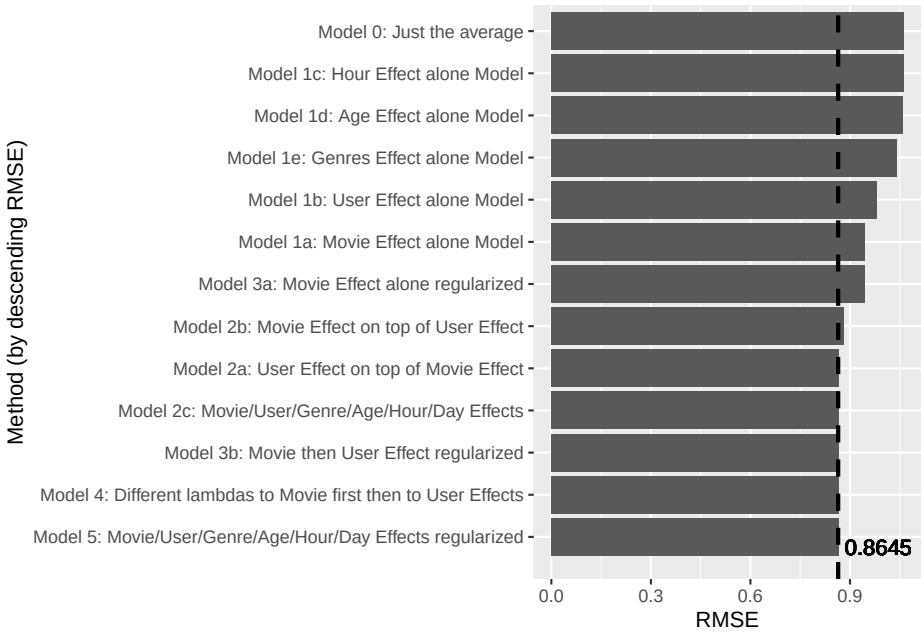
```
#building of the final model with the edx set and final lambda

mu <- mean(edx$rating)

movie_avgs <- edx %>%
  group_by(movieId) %>%
```

```
  summarize(b_i = sum(rating - mu)/(n()+lambda_final))

user_avgs <- edx %>%
  left_join(movie_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda_final))

genre_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+lambda_final))

age_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(movie_age) %>%
  summarize(b_a = sum(rating - mu - b_i - b_u - b_g)/(n()+lambda_final))

hour_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(age_avgs, by='movie_age') %>%
  group_by(hour) %>%
  summarize(b_h = sum(rating - mu - b_i - b_u - b_g - b_a)/(n()+lambda_final))

weekday_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(age_avgs, by='movie_age') %>%
  left_join(hour_avgs, by='hour') %>%
  group_by(weekday) %>%
  summarize(b_w = sum(rating - mu - b_i - b_u - b_g - b_a - b_h)/(n()+lambda_final))

predicted_ratings <- validation %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(age_avgs, by='movie_age') %>%
  left_join(hour_avgs, by='hour') %>%
  left_join(weekday_avgs, by='weekday') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_a + b_h + b_w) %>%
  .$pred

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

```
## [1] 0.8644685
```

The final RMSE obtained on the "validation" set with "Model 5" is 0.8644685 (represented with a dashed

line in the below summary plot) which is in fact even lower than the result for "Model 5" when evaluating on the "edx" test set for comparing across models.




# 6 Additional analysis: Matrix Factorization

## 6.1 About "recosystem" package

As an additional exercise, a test has been performed using "recosystem" package. This easy-to-use package performs matrix factorization specifically for recommender systems, specifically for settings in which the matrix A has many missing values.

The package provides a number of user-friendly R functions to simplify data processing and model building. Also, unlike most other R packages for statistical modeling, "recosystem" stores data in the hard disk rather than in memory achieving great performance.

The aim of matrix factorization is to approximate the whole rating matrix $R_{m\,x\,n}$ by the product of two matrices of lower dimensions, $P_{k\,x\,m}$ and $Q_{k\,x\,n}$, such that:

$$R \approx P\prime Q$$

Then $p_u$ will be the $u$-th column of $P$, and $q_v$ be the $v$-th column of $Q$, then the rating given by user $u$ on item $v$ would be predicted as $p\prime_u q_v$, this will allow to fill the blanks in the ratings matrix.

In order to fulfill this task, the algorithm will iterate to find the best approximation for $P$ and $Q$.

Full detail and examples of "recosystem" package can be found in: https://cran.r-project.org/web/packages/recosystem/recosystem.pdf

## 6.2 Applying Matrix Factorization

Following the same analysis workflow to train the model on "edx" train partition and evaluate on "edx" test dataset, after executing the following code an RMSE of 0.7863502 already shows big improvement compared to regression techniques.

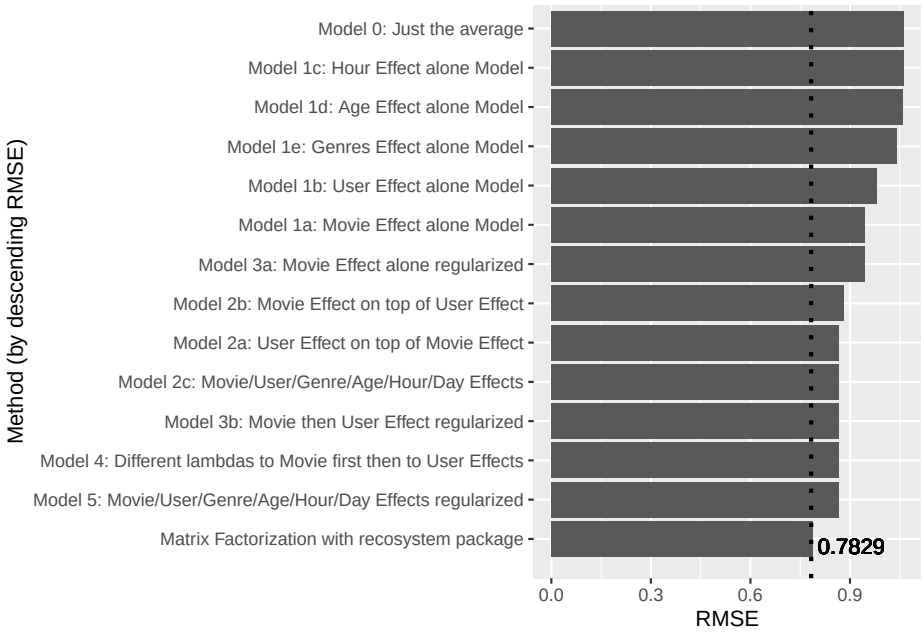
| method | RMSE |
|---|---|
| Model 0: Just the average | 1.0611350 |
| Model 1a: Movie Effect alone Model | 0.9441568 |
| Model 1b: User Effect alone Model | 0.9795916 |
| Model 1c: Hour Effect alone Model | 1.0611016 |
| Model 1d: Age Effect alone Model | 1.0601794 |
| Model 1e: Genres Effect alone Model | 1.0411427 |
| Model 2a: User Effect on top of Movie Effect | 0.8659736 |
| Model 2b: Movie Effect on top of User Effect | 0.8826201 |
| Model 2c: Movie/User/Genre/Age/Hour/Day Effects | 0.8655933 |
| Model 3a: Movie Effect alone regularized | 0.9441164 |
| Model 3b: Movie then User Effect regularized | 0.8654678 |
| Model 4: Different lambdas to Movie first then to User Effects | 0.8654669 |
| Model 5: Movie/User/Genre/Age/Hour/Day Effects regularized | 0.8651046 |
| Matrix Factorization with recosystem package | 0.7867029 |

For simplification purposes here's the code to prepare data in a friendly format for "recosystem" package and create the model on the entire "edx" set to later evaluate the final results on the "validation set" as previously done.

```
#building the model on the edx entire set

edx_data <-  with(edx, data_memory(user_index = userId,
                                   item_index = movieId,
                                   rating = rating))
validation_data  <-  with(validation, data_memory(user_index = userId,
                                                  item_index = movieId,
                                                  rating = rating))
r <-  recosystem::Reco()

opts <-  r$tune(edx_data, opts = list(dim = c(10, 20, 30),
                                      lrate = c(0.1, 0.2),
                                      costp_l2 = c(0.01, 0.1),
                                      costq_l2 = c(0.01, 0.1),
                                      nthread  = 4, niter = 10))

r$train(edx_data, opts = c(opts$min, nthread = 4, niter = 20))

#applying the model to the validation data set
```

```
predicted_ratings <-  r$predict(validation_data, out_memory())

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

```
## [1] 0.7829408
```




The RMSE results in 0.7822679 (illustrated with a dotted line) considerably beating the previous techniques, and even the result of "Model 5".

# 7 Final conclusions

With simple regression techniques improved by regularisation, the project target has been achieved.

In practical effects, the addition of some extra effects might not be of use if the prediction exercise has to be ran on a daily or weekly basis, as opposed of it being "real-time".

Additionally, as suggested by the course literature on the "Netflix" challenge experience, other techniques such as matrix factorization can radically improve the results of the exercise, but comes at expenses of long calculation times.

Further ideas to explore can leverage on *knn* techniques to apply movie recommendation ratings on similar users, or also developing derivated predictors based on more mutations of the original data.