Movie recommendation using the MovieLens dataset

Felipe Urrego

02/03/2019

Introduction

- What is Machine Learning?

In simple words, we can say that it is just a problem of approximation of a function or a relationship with the purpose to obtain new information. It can be any function of any complexity. It can even do not exist, in this case, we assume that there is a kind of law of nature or a relationship that we want to approximate.

- Notations

We note this function $f(x)$ and its values $y = f(x)$ (or $y = f(x) + \epsilon$ where ϵ is random). Conventions for x and y depend on the field of science - Machine Learning or Statistical Learning - they are a bit different.

x / Features / Independent Variables (it's not the case for the raw data, but we should treat it in order to get only independent variables, for example, with help of PCA - Principal Component Analysis). y / Label / Dependent Variable.

x can be a vector (vector of features), its length is the number of independent variables. y is usually a scalar value.

Observation is one couple $\{x, y\}$ or just one x.

There are different types of Machine Learning problems, but we now consider Supervised Learning problem, it is when we have labels as described.

- Problem formulation

We have two sets of data, one is labelled (it has x and y and called 'train/training set') and another not (it has only x and called 'test set'). Train set is $\{x_i, y_i\}_{i=0,1..n}$, where n is the number of observations (they ideally should be independent).

The objective is to get y for the test set (we say to predict or to forecast). So, we choose a Machine Learning model g and train (or fit) it on the train set (calibrate its internal parameters) so that $g(x)$ looks similar to y for $\{x, y\}$ in the train set. We quantify "look similar" by introducing a function to minimise called Error Measure. Thus, g is our approximation of f and $g(x)$ for x in the test set (called Prediction) is our guess of y for the test set as asked.

- Error measure

There are different error measures (also called Loss/Cost/Objective Functions), but here we use the most common one - Root Mean Square Error (RMSE), for a model g:

$$RMSE(g) = \sqrt{\frac{1}{n}\sum_{i=1..n}(y_i - g(x_i))^2}$$

```
RMSE <- function(y, g)
  return(sqrt(mean( (y-g)**2 )))
```

- Overfitting

Almost always the data we have is not a pure reflection of a law of nature but also of some noises (it is why it is hard to find $g \equiv f$) and if we consider a model too complicated for the problem, it can also fit the noise, as

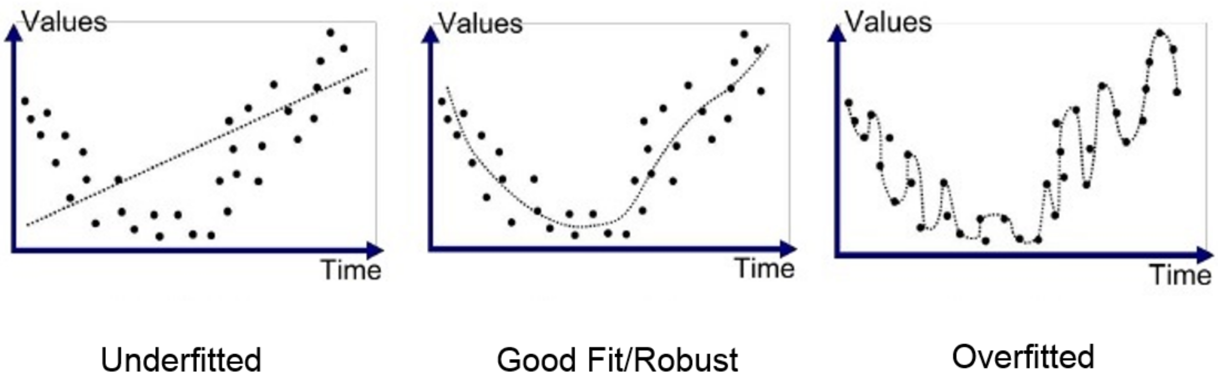


Figure 1: Overfitting

on the figure below. A model that is too simple is not a good choice neither. In order to find a suitable model from a set of models, we need to test models on data that was not used in the training. For this purpose we divide the train set into two parts (80% and 20%), then we train each model on 80% data and compare its predictions over the 20% data with the real labels. It gives a RMSE per model, and a model with the smallest RMSE is the winner.

- MovieLens

In the current project, we consider the MovieLens 10M database, it consists of movies and users ratings and we want to be able to say which rating a specific user would give to a specific movie (in order to know what to propose to watch to this user). It consists of 10 million of ratings (10 M rows), for each rating we have a user ID, a time when the rating was given and a movie ID. Each movie has a title (equivalent to movie ID) with the year of production, an and a set of genres (a movie can have multiple genres).

Preprocessing

```
# List of packages we need
list_of_packages <- list("randomForest",
                         "kableExtra",
                         "tidyverse",
                         "lubridate",
                         "corrplot",
                         "ggplot2",
                         "readxl",
                         "dslabs",
                         "knitr",
                         "mlr")

# Function that loads and installs if necessary indicated packages
UsePackages = function(list_of_packages) {
  for (p in list_of_packages){
    # if (!is.element(p, installed.packages()[,1]))
    #   install.packages(p)
    require(p, character.only = TRUE)
  }
```

```r
}
UsePackages(list_of_packages)

# precision
prec <- 3

# data frame to stock results
results <- tibble()
```

Create test and validation sets

Create edx set (train set) and validation set (test and validatiob set in the same time)

```r
# Note: this process could take a couple of minutes

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

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


# Avoid downloading data we already have
if (file.exists("ml-10M100K/ratings.dat") & file.exists("ml-10M100K/movies.dat")){

  ratings <- read.table(text = gsub("::", "\t", readLines("ml-10M100K/ratings.dat")),
                        col.names = c("userId", "movieId", "rating", "timestamp"))
  movies <- str_split_fixed(readLines("ml-10M100K/movies.dat"), "\\::", 3)

}else{

  dl <- tempfile()
  download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
  ratings <- read.table(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                        col.names = c("userId", "movieId", "rating", "timestamp"))
  movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)

}

# Treat data
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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


# Validation set will be 10% of MovieLens data
set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
```

```r
edx <- movielens[-test_index,]
temp <- movielens[test_index,]


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


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

rm(dl, ratings, movies, test_index, temp, movielens, removed)

# change the format from 'data.frame' to 'tibble'. And blend the rows
edx <- as_tibble(edx)[sample(1:nrow(edx)),]
validation <- as_tibble(validation)[sample(1:nrow(validation)),]
```

Feature Selection and Feature Engineering

Feature Selection means that we remove irrelevant variables that only add noise.

Feature Engineering means that we add new variables.

Year of production and year of rating

```r
# extract year from the title and remove title
edx <- edx %>%
  extract(title, "year", regex="\\(([0-9 \\-]*)\\)$") %>% mutate(year=as.integer(year))
validation <- validation %>%
  extract(title, "year", regex="\\(([0-9 \\-]*)\\)$") %>% mutate(year=as.integer(year))

# timestamp to year of the publication of rating
edx <- edx %>%
  mutate(timestamp = as.integer(year(as_datetime(timestamp))))
validation <- validation %>%
  mutate(timestamp = as.integer(year(as_datetime(timestamp))))
```

Add mean/median rating per year

```r
df_years <- edx %>% group_by(year) %>%
  summarise(mean_per_year=mean(rating), median_per_year=median(rating))
edx <- edx %>% left_join(df_years, by="year")
```

Genres

```r
# data frame of genres and its numbers
(df_genres <- edx %>%
```

```
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(number = n(), mean_rating = mean(rating)) %>%
  arrange(desc(number))
```

```
## # A tibble: 20 x 3
##    genres               number mean_rating
##    <chr>                 <int>       <dbl>
##  1 Drama               3910127        3.67
##  2 Comedy              3540930        3.44
##  3 Action              2560545        3.42
##  4 Thriller            2325899        3.51
##  5 Adventure           1908892        3.49
##  6 Romance             1712100        3.55
##  7 Sci-Fi              1341183        3.40
##  8 Crime               1327715        3.67
##  9 Fantasy              925637        3.50
## 10 Children             737994        3.42
## 11 Horror               691485        3.27
## 12 Mystery              568332        3.68
## 13 War                  511147        3.78
## 14 Animation            467168        3.60
## 15 Musical              433080        3.56
## 16 Western              189394        3.56
## 17 Film-Noir            118541        4.01
## 18 Documentary           93066        3.78
## 19 IMAX                   8181        3.77
## 20 (no genres listed)        7        3.64
```

Divide 'genres' into separate genres

```
#' create a data.frame with dummy genres columns from a list of mixed genres
GetDummyGenres <- function(my_vector, my_genres){
  df2 <- sapply(my_vector,
              function(x){
                zeros <- rep(0,length(my_genres))
                x <- strsplit(x, "\\|")[[1]] # split by char "|" into two strings
                zeros[match(x, my_genres)] <- 1
                return(as.integer(zeros))
              },
              USE.NAMES=FALSE) %>% t
  colnames(df2) <- my_genres
  df2 <- df2 %>% as_tibble # %>% select(-`(no genres listed)`)# %>% mutate_all(as.factor)
  return(df2)
}

# movies and its genres
df_movies <- edx %>% group_by(movieId) %>%
  summarise("mean_per_movie"=mean(rating),
            "median_per_movie"=median(rating),
            "number"=n(),
            "genres" = genres[1])
df_movies <- df_movies %>% bind_cols(GetDummyGenres(.$genres, df_genres$genres))
```

A movie can have multiple genres, so can estimate its rating as average of average ratings per genres

```
(df_movies <- df_movies %>%
  mutate(mean_per_genre =
           rowSums(as.matrix(df_movies %>% select(df_genres$genres)) * df_genres$mean_rating) /
           rowSums(df_movies %>% select(df_genres$genres))) %>%
   select(mean_per_genre, names(df_movies)))
```

```
## # A tibble: 10,677 x 26
##    mean_per_genre movieId mean_per_movie median_per_movie number genres
##             <dbl>   <dbl>          <dbl>            <dbl>  <int> <chr>
##  1           3.67       1           3.93                4  23790 Adven~
##  2           3.59       2           3.21                3  10779 Adven~
##  3           3.65       3           3.15                3   7028 Comed~
##  4           3.56       4           2.86                3   1577 Comed~
##  5           3.44       5           3.07                3   6400 Comedy
##  6           3.72       6           3.82                4  12346 Actio~
##  7           3.59       7           3.36                3   7259 Comed~
##  8           3.61       8           3.13                3    821 Adven~
##  9           3.42       9           3.00                3   2278 Action
## 10           3.65      10           3.43                3  15187 Actio~
## # ... with 10,667 more rows, and 20 more variables: Drama <int>,
## #   Comedy <int>, Action <int>, Thriller <int>, Adventure <int>,
## #   Romance <int>, `Sci-Fi` <int>, Crime <int>, Fantasy <int>,
## #   Children <int>, Horror <int>, Mystery <int>, War <int>,
## #   Animation <int>, Musical <int>, Western <int>, `Film-Noir` <int>,
## #   Documentary <int>, IMAX <int>, `(no genres listed)` <int>
```

```
edx <- edx %>% left_join(df_movies %>% select(movieId, mean_per_genre), by="movieId")
```

Add mean/median rating per movie

```
edx <- edx %>%
  left_join(df_movies %>% select(movieId, mean_per_movie, median_per_movie), by="movieId")
```

Add mean/median rating per user

```
df_users <- edx %>% group_by(userId) %>% summarise("mean_per_user"=mean(rating),
                                                  "median_per_user"=median(rating),
                                                  "number"=n())
edx <- edx %>% left_join(df_users %>% select(-number), by="userId")
```

Data Summary and Data Visualisation

```
# summary
print(summary(edx))
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :1995
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:2000
##  Median :35738   Median : 1834   Median :4.000   Median :2002
```

```
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :2002
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:2005
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :2009
##       year         genres          mean_per_year   median_per_year
##  Min.   :1915   Length:9000055     Min.   :3.285   Min.   :3.000
##  1st Qu.:1987   Class :character   1st Qu.:3.431   1st Qu.:3.500
##  Median :1994   Mode  :character   Median :3.460   Median :3.500
##  Mean   :1990                      Mean   :3.512   Mean   :3.589
##  3rd Qu.:1998                      3rd Qu.:3.530   3rd Qu.:3.500
##  Max.   :2008                      Max.   :4.053   Max.   :4.000
##  mean_per_genre  mean_per_movie  median_per_movie mean_per_user
##  Min.   :3.270   Min.   :0.500   Min.   :0.500    Min.   :0.500
##  1st Qu.:3.514   1st Qu.:3.218   1st Qu.:3.000    1st Qu.:3.252
##  Median :3.581   Median :3.591   Median :4.000    Median :3.529
##  Mean   :3.585   Mean   :3.512   Mean   :3.598    Mean   :3.512
##  3rd Qu.:3.649   3rd Qu.:3.876   3rd Qu.:4.000    3rd Qu.:3.800
##  Max.   :4.012   Max.   :5.000   Max.   :5.000    Max.   :5.000
##  median_per_user
##  Min.   :0.500
##  1st Qu.:3.000
##  Median :4.000
##  Mean   :3.607
##  3rd Qu.:4.000
##  Max.   :5.000
```

```r
# check if there are NA in data
cat("Number of rows containing NA :", edx %>% filter(!complete.cases(.)) %>% nrow, "\n")
```

```
## Number of rows containing NA : 0
```

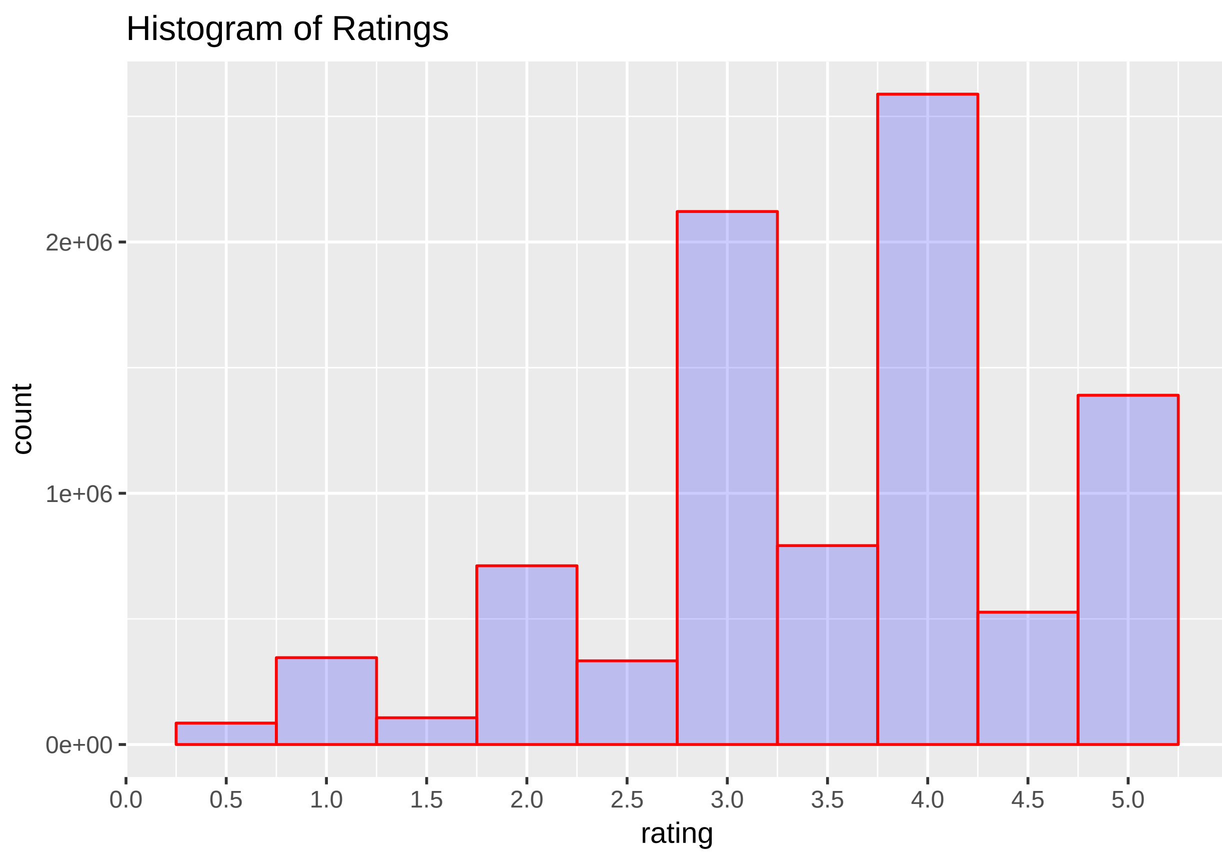
```r
cat("Number of users  :", nrow(df_users), "\n")
```

```
## Number of users  : 69878
```

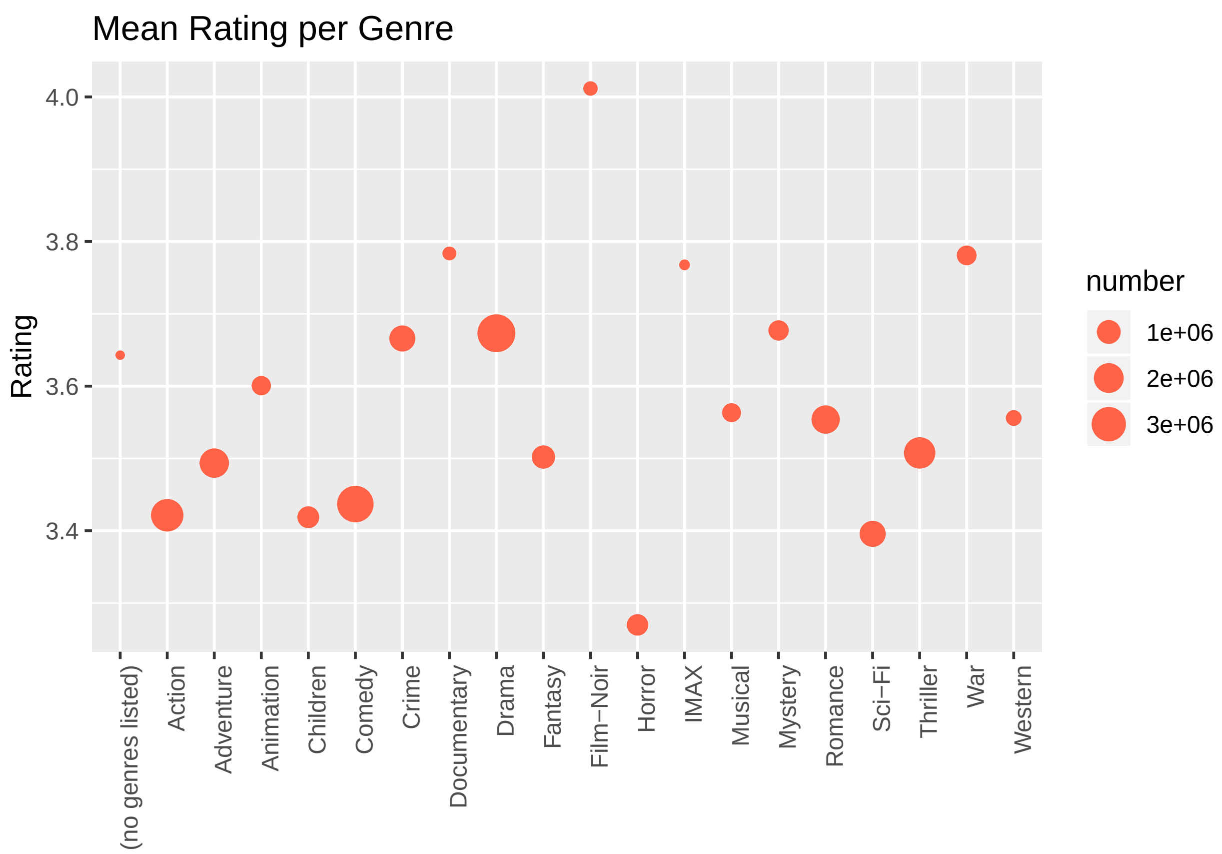
```r
cat("Number of movies :", nrow(df_movies), "\n")
```

```
## Number of movies : 10677
```

```r
# plot a Histogram of Ratings
ggplot(edx, aes(rating)) +
  geom_histogram(binwidth=0.5, fill=I("blue"), col=I("red"), alpha=I(.2)) +
  ggtitle("Histogram of Ratings") +
  scale_x_continuous(breaks=seq(0,5,.5))
```



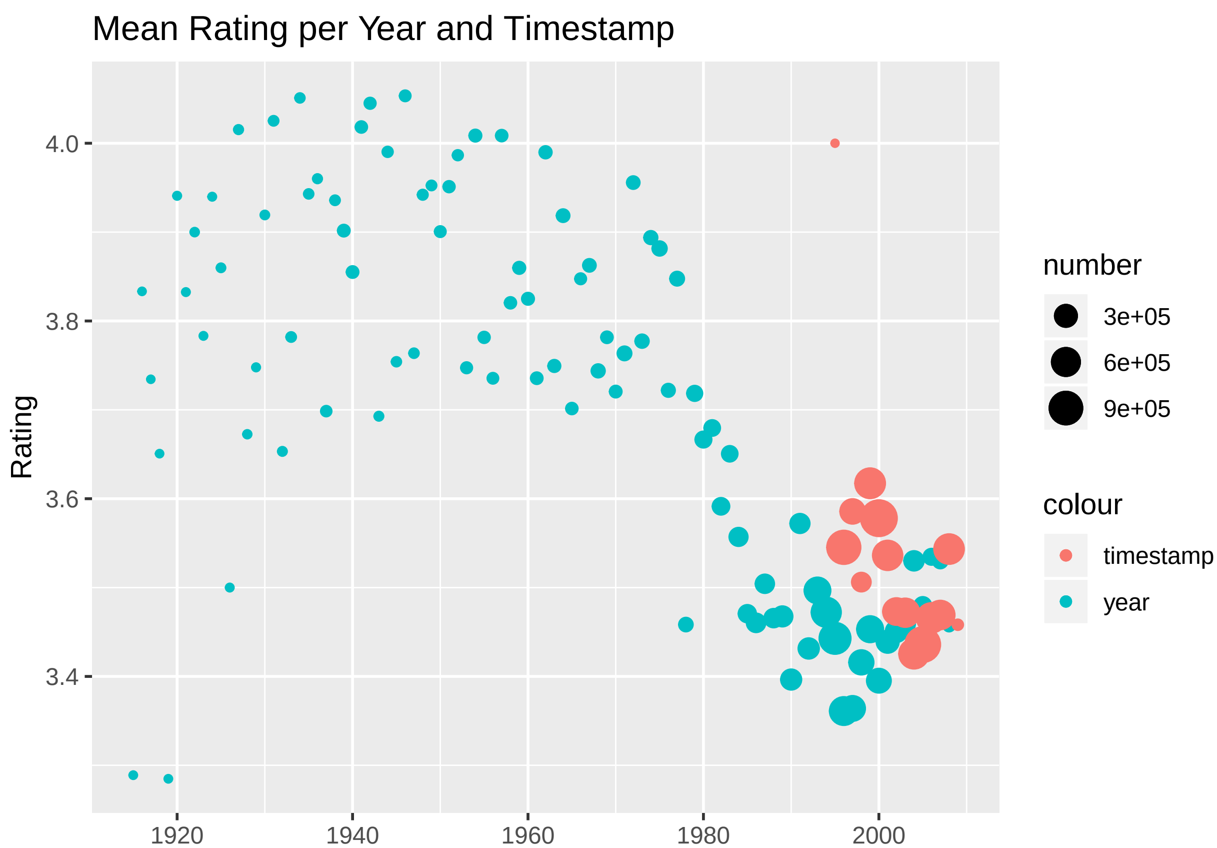

```
# plot Mean Rating per Genre
ggplot(df_genres) + ggtitle("Mean Rating per Genre") + xlab(NULL) + ylab("Rating") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  geom_point(aes(genres, mean_rating, size=number), col="tomato")
```



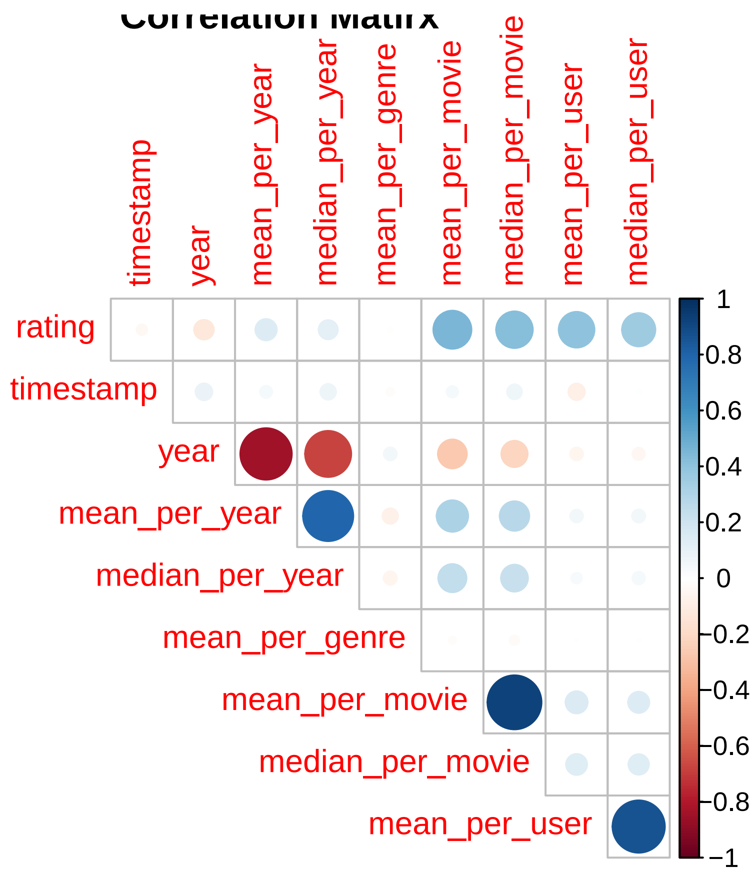

```
cat("We clearly see that the number of movies produced per year increases with time while the mean ratin
```

```
## We clearly see that the number of movies produced per year increases with time while the mean rating
```

```
# plot Mean Rating per Year and Timestamp
ggplot() +
  geom_point(data = edx %>% group_by(year) %>% summarise(mean_rating = mean(rating), number=n()),
             aes(year, mean_rating, col="year", size=number)) +
  geom_point(data = edx %>% group_by(timestamp) %>% summarise(mean_rating = mean(rating), number=n()),
             aes(timestamp, mean_rating, col="timestamp", size=number)) +
  ggtitle("Mean Rating per Year and Timestamp") + xlab(NULL) + ylab("Rating")
```



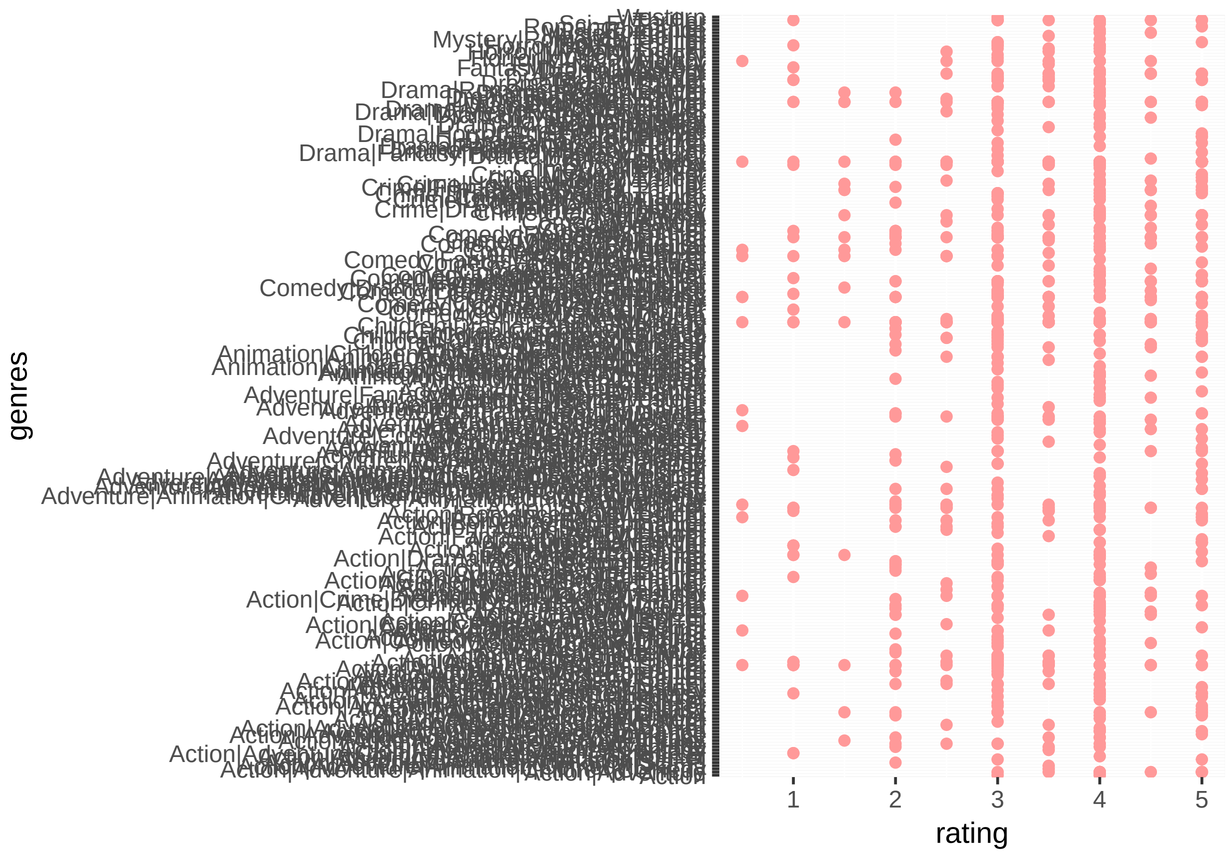

```
# plot correlations
corrplot(cor(edx %>% select(-c(genres, movieId, userId))), type="upper", diag=FALSE, title="Correlation
```



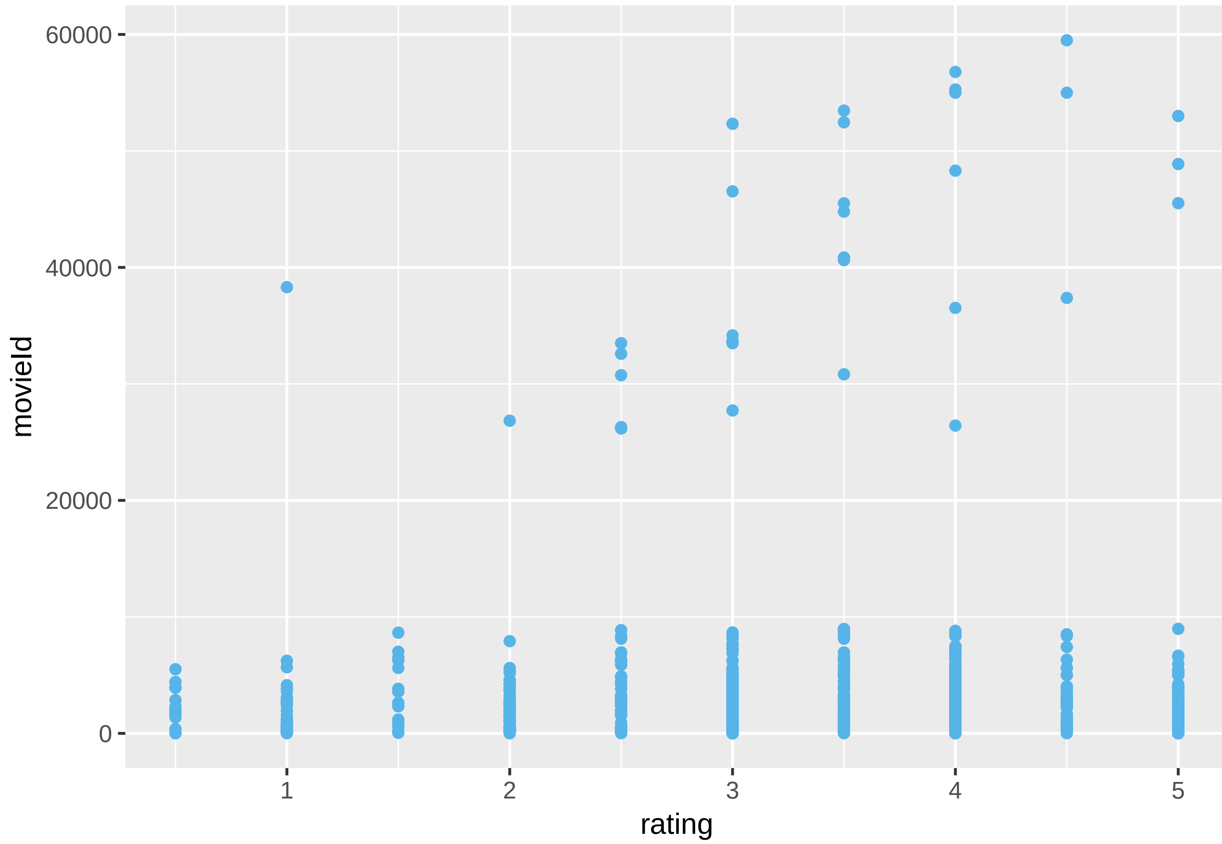

```
cat("We see that the rating is quite correlated with mean per user and mean per movie which are in thei
```

```
## We see that the rating is quite correlated with mean per user and mean per movie which are in their
```

```
# edx is too big to plot it all, so the next analysis is done over a small part of edx
edx_short <- edx[1:1000,]

# plot ratings vs other variables
ggplot(edx_short) + geom_point(aes(rating, genres), col="#FF9999")
```



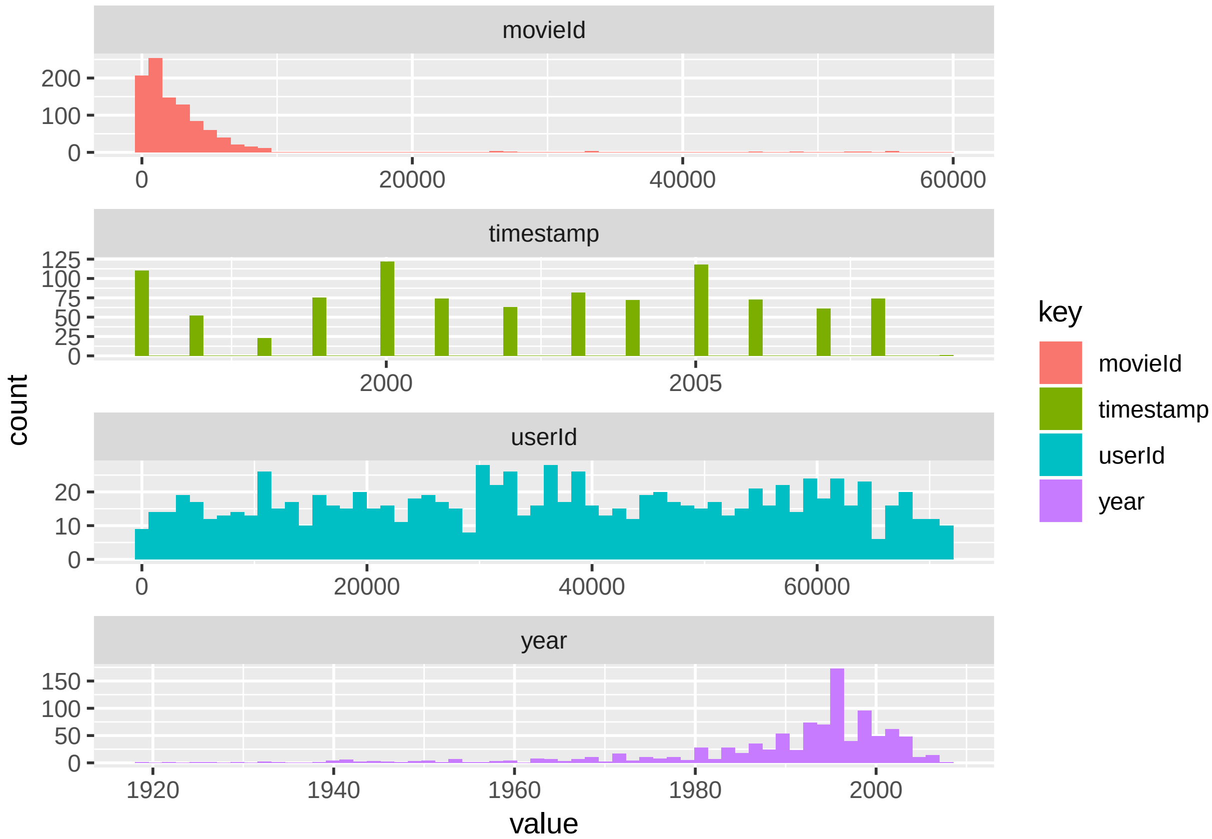

```r
ggplot(edx_short) + geom_point(aes(rating, movieId), col="#56B4E9")
```




```
cat("we can see that some ratings are less frequant for some generes\n")
```

```
## we can see that some ratings are less frequant for some generes
```

```
# plot histrograms
# df <- gather(edx_short %>% select(-c("genres", "timestamp")))
df <- gather(edx_short %>% select(userId, movieId, timestamp, year))
ggplot(df, aes(value, fill=key)) +
  facet_wrap(~key, scales="free", ncol=1) +
  geom_histogram(bins=60)
```




```
cat("We see that our data is not homogeneous\n")
```

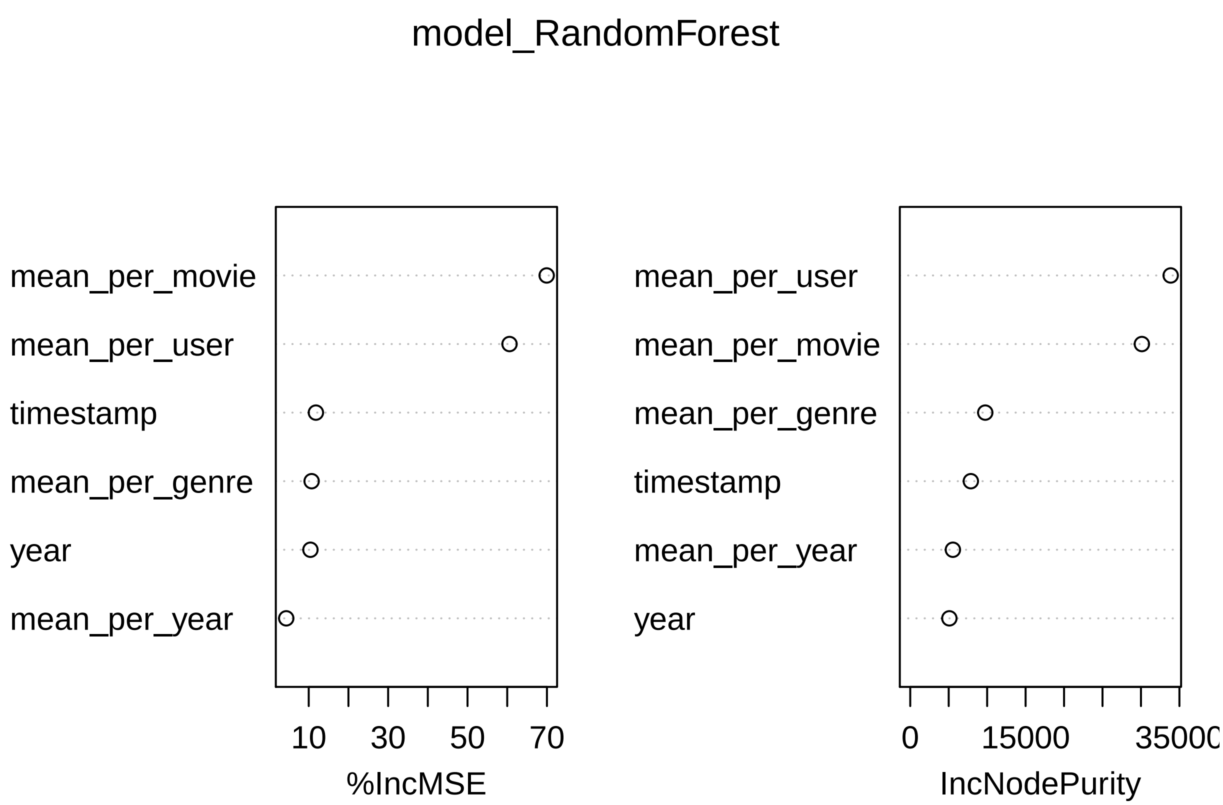
```
## We see that our data is not homogeneous
```

```
# Conclusion
cat("We do not see any obvious pattern that would certainly help us to determine a rating\n")
```

```
## We do not see any obvious pattern that would certainly help us to determine a rating
```

Feature Importance

```
model_RandomForest <- randomForest(rating ~ .,
                                   data = edx[1:1e5,] %>%
                                     select(-c(userId, movieId, genres, contains("median"))),
                                   ntree=10, keep.forest=FALSE, importance=TRUE)
varImpPlot(model_RandomForest)
```



Model 0: Mean/Median Rating

```
# get mean and median and calculate rmse
rating_mean   <- mean(edx$rating)
rating_median <- median(edx$rating)
cat("Mean Rating:  ", round(rating_mean, prec),"\n")
```

```
## Mean Rating:   3.512
```

```
cat("Median Rating:", round(rating_median, prec),"\n")
```

```
## Median Rating: 4
```

```
results_temp <- tibble("Model" = c("mean", "median"),
                       "RMSE"  = c(RMSE(validation$rating, rating_mean),
                                   RMSE(validation$rating, rating_median)))
# add it to results base
results <- results %>% bind_rows(results_temp)
# show results table in latex format
kable(results_temp, "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

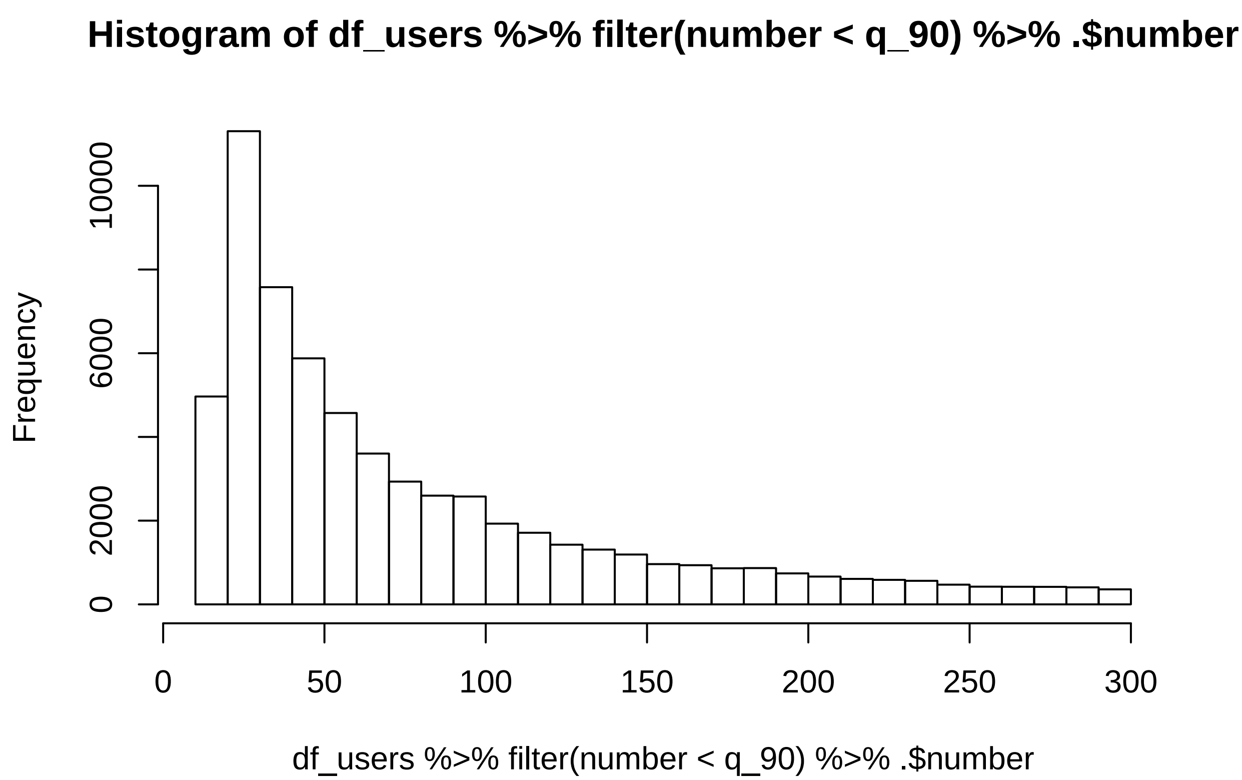
Model	RMSE
mean	1.061202
median	1.168016

Model 1: Mean/Median Rating per User

```r
cat(min(df_users$number), max(df_users$number), "\n")
```

```
## 10 6616
```

```r
q_90 <- quantile(df_users$number, 0.9)
hist(df_users %>% filter(number<q_90) %>% .$number, breaks = 30)
```




```r
# prediction
validation <- validation %>% left_join(df_users %>% select(-number), by="userId")

# get rmse and stock it
results_temp <- tibble("Model" = c("mean_per_user", "median_per_user"),
                       "RMSE"  = c(RMSE(validation$rating, validation$mean_per_user),
                                   RMSE(validation$rating, validation$median_per_user)))
results <- results %>% bind_rows(results_temp)
# show results table in latex format
kable(results_temp, "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

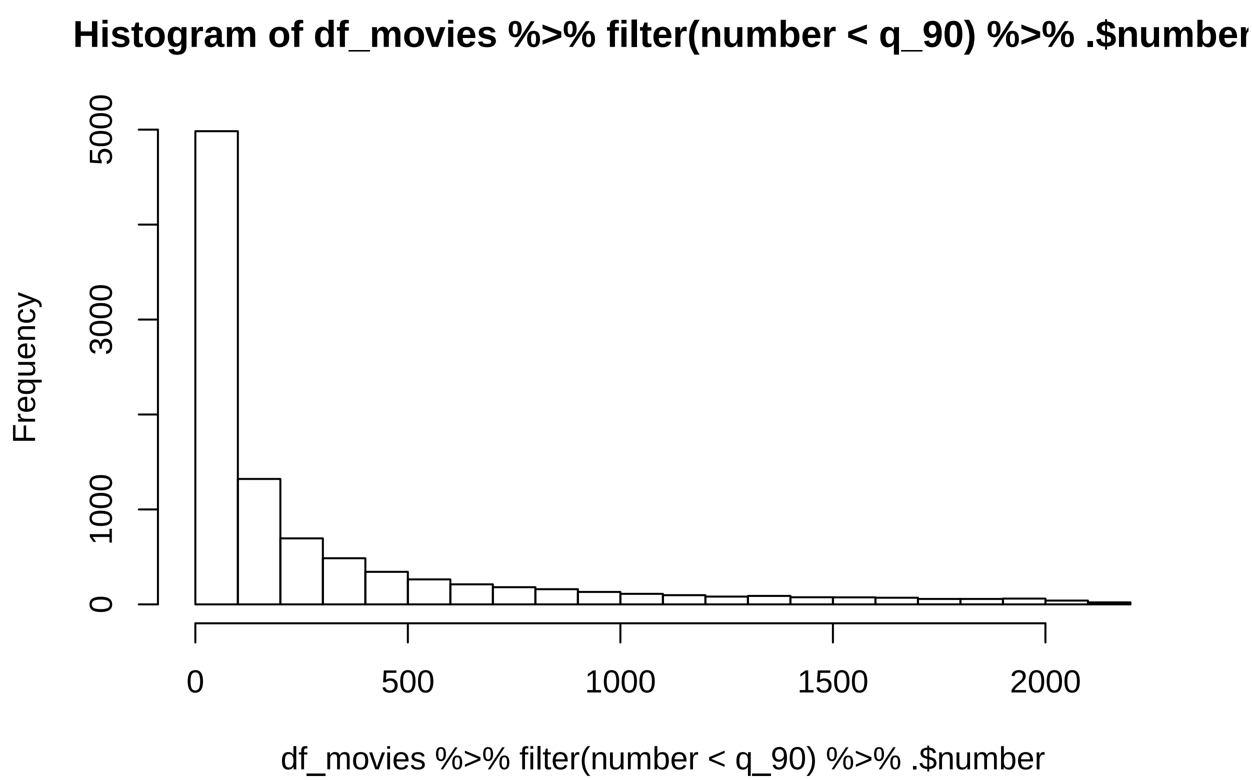
Model	RMSE
mean_per_user	0.978336
median_per_user	1.021136

Model 2: Mean/Median Rating per Movie

```r
cat(min(df_movies$number), max(df_movies$number), "\n")
```

```
## 1 31362
```

```r
q_90 <- quantile(df_movies$number, 0.9)
hist(df_movies %>% filter(number<q_90) %>% .$number, breaks = 30)
```




```r
# prediction
validation <- validation %>%
  left_join(df_movies %>% select(movieId, mean_per_movie, median_per_movie), by="movieId")

# get rmse and stock it
results_temp <- tibble("Model" = c("mean_per_movie", "median_per_movie"),
                       "RMSE"  = c(RMSE(validation$rating, validation$mean_per_movie),
                                   RMSE(validation$rating, validation$median_per_movie)))
results <- results %>% bind_rows(results_temp)
# show results table in latex format
kable(results_temp, "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

Model	RMSE
mean_per_movie	0.9439087
median_per_movie	0.9716910

Model 3 Mean/Median Rating per Year

```r
# prediction
validation <- validation %>% left_join(df_years, by="year")

# get rmse and stock it
results_temp <- tibble("Model" = c("mean_per_year", "median_per_year"),
                       "RMSE"  = c(RMSE(validation$rating, validation$mean_per_year),
                                   RMSE(validation$rating, validation$median_per_year)))
results <- results %>% bind_rows(results_temp)
# show results table in latex format
kable(results_temp, "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

Model	RMSE
mean_per_year	1.050026
median_per_year	1.066137

Model 4 Mean Rating per Genre

```r
# prediction
validation <- validation %>% left_join(df_movies %>% select(movieId, mean_per_genre), by="movieId")

# get rmse and stock it
results_temp <- tibble("Model" = "mean_per_genre",
                       "RMSE"  = RMSE(validation$rating, validation$mean_per_genre))
results <- results %>% bind_rows(results_temp)
# show results table in latex format
kable(results_temp, "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

Model	RMSE
mean_per_genre	1.070248

Model 5 Ordinary Least Squares regression (OLS)

```r
# data for regression type train
edx2 <- edx %>% select(-c(userId, movieId, genres, contains("median")))
```

```r
ols <- lm(rating ~ ., data = edx2)
summary(ols)
```

```
##
## Call:
## lm(formula = rating ~ ., data = edx2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.6629 -0.4994  0.0693  0.5856  4.8641
```

```
##
## Coefficients:
##                  Estimate Std. Error  t value Pr(>|t|)
## (Intercept)     8.458e+00  1.671e-01   50.622   <2e-16 ***
## timestamp      -7.366e-03  8.100e-05  -90.948   <2e-16 ***
## year            1.659e-03  4.063e-05   40.831   <2e-16 ***
## mean_per_year   1.245e-01  3.671e-03   33.925   <2e-16 ***
## mean_per_genre -1.807e-03  2.642e-03   -0.684    0.494
## mean_per_movie  8.884e-01  6.366e-04 1395.644   <2e-16 ***
## mean_per_user   8.399e-01  6.894e-04 1218.263   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8714 on 9000048 degrees of freedom
## Multiple R-squared:  0.3246, Adjusted R-squared:  0.3246
## F-statistic: 7.21e+05 on 6 and 9000048 DF,  p-value: < 2.2e-16
```

```r
# test
results_temp <- tibble("Model" = "ols",
                       "RMSE"  = RMSE(validation$rating, predict(ols, validation)))
results <- results %>% bind_rows(results_temp)
# show results table in latex format
kable(results_temp, "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

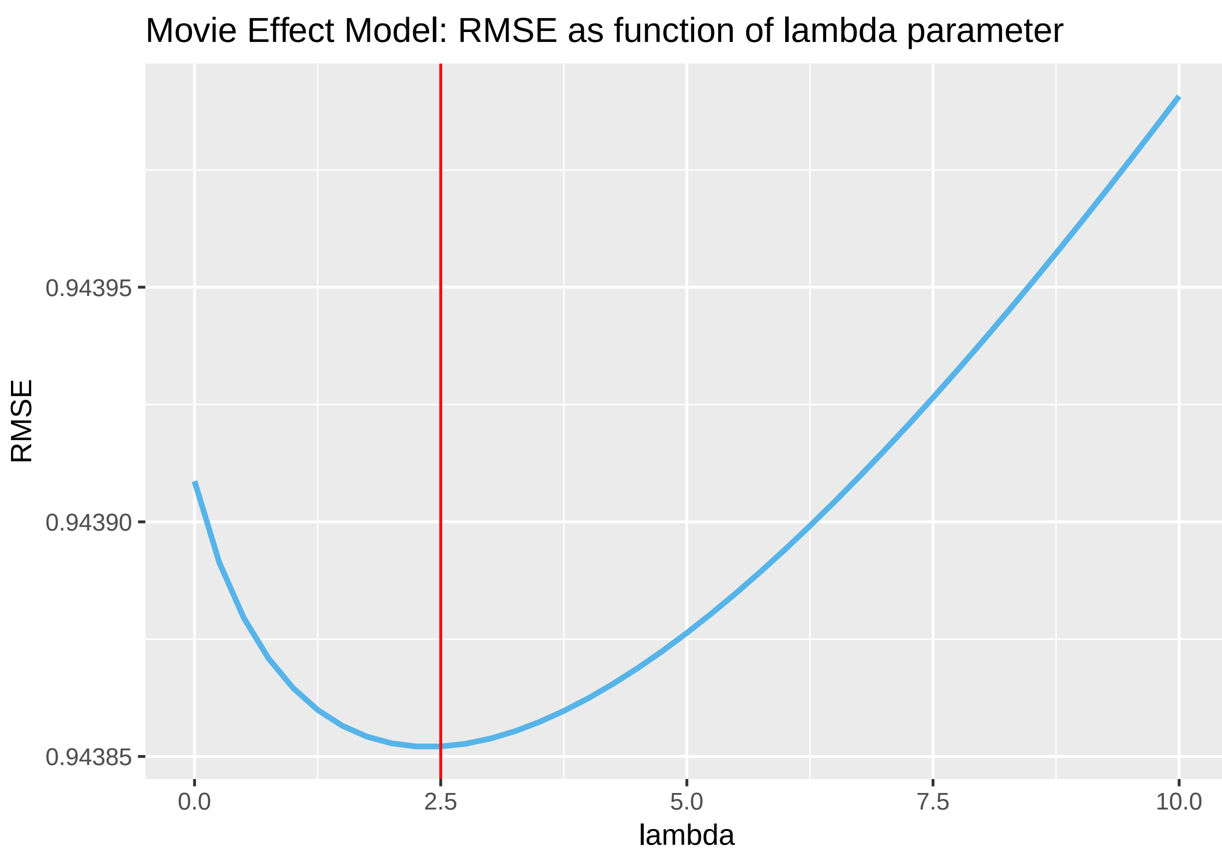
Model	RMSE
ols	0.8786586

Model 6 Movie Effect Model

$$y_i = \mu + b_{movie(i)} + \epsilon_i$$

```r
Movie_Effect_Model <- function(lambda, return_prediction=FALSE){
  # the average rating
  mu <- mean(edx$rating)

  # calculate b_movie coefficients
  movie_effect <- edx %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu) / (n() + lambda))

  # add to validation
  validation <- validation %>% left_join(movie_effect, by='movieId')

  # rmse
  my_rmse <- RMSE(validation$rating, (mu + validation$b_m))

  if (return_prediction){
    return(list("rmse"       = my_rmse,
                "prediction" = mu + validation$b_m))
  }else{
    return(my_rmse)
  }
```

```r
}

# calculate errors for a set of lambda values and choose the smallest rmse
lambdas <- seq(0, 10, 0.25)
model_rmses <- sapply(lambdas, Movie_Effect_Model)
lambda_of_smallest_rmse <- lambdas[which.min(model_rmses)]

# plot it
ggplot() + geom_line(aes(lambdas, model_rmses), col="#56B4E9", size=1) +
  geom_vline(xintercept=lambda_of_smallest_rmse, col="red") + ggtitle("Movie Effect Model: RMSE as func
```



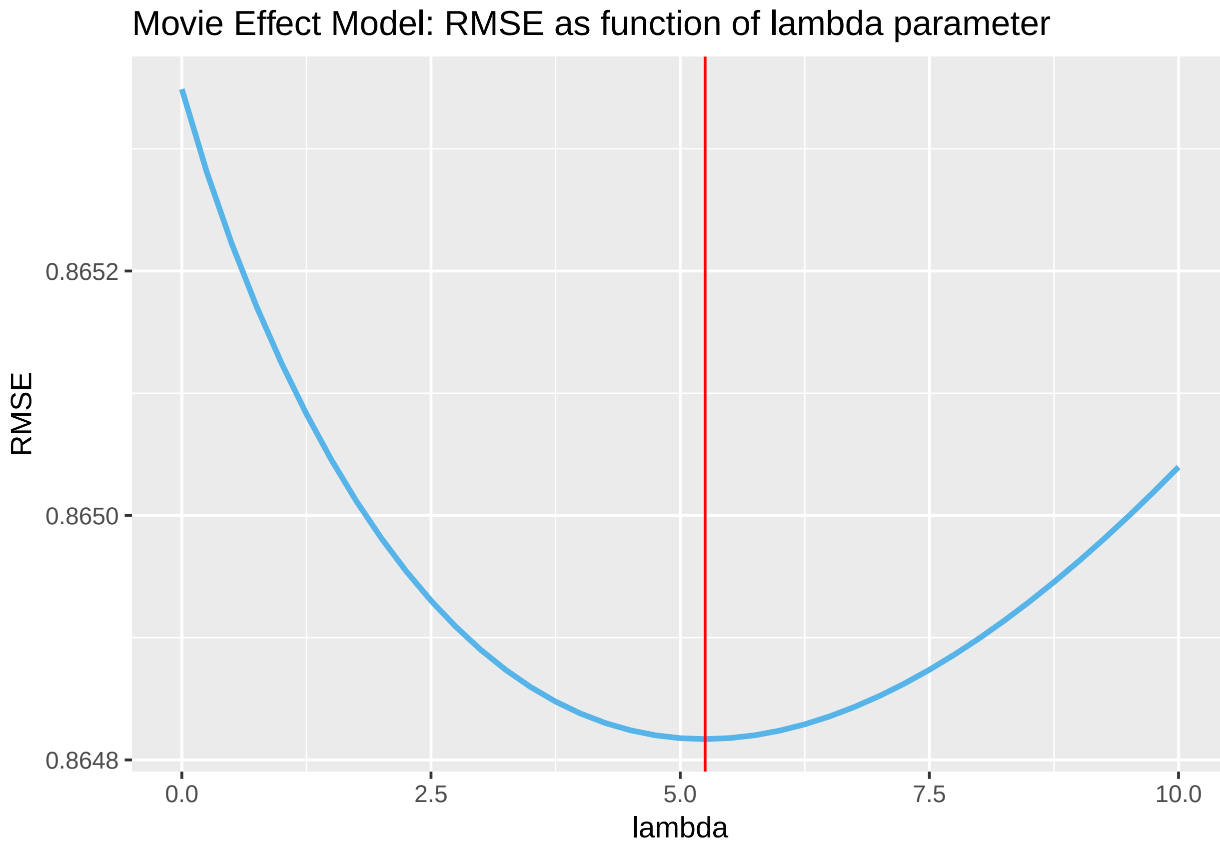

```r
# add the best prediction
results_temp <- tibble("Model" = "movie_effect",
                       "RMSE"  = model_rmses[which.min(model_rmses)])
results <- results %>% bind_rows(results_temp)
# show results table in latex format
kable(results_temp, "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

Model	RMSE
movie_effect	0.9438521

Model 7 Movie User Effect Model

$$y_i = \mu + b_{movie(i)} + b_{user(i)} + \epsilon_i$$

```
Movie_User_Effect_Model <- function(lambda, return_prediction=FALSE){
  # the average rating
  mu <- mean(edx$rating)

  # calculate b_movie coefficients
  movie_effect <- edx %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu) / (n() + lambda))

  # calculate b_user coefficients
  movie_user_effect <- edx %>%
    left_join(movie_effect, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_m) / (n() + lambda))

  # add to validation
  validation <- validation %>%
    left_join(movie_effect, by='movieId') %>%
    left_join(movie_user_effect, by='userId')

  # rmse
  my_rmse <- RMSE(validation$rating, (mu + validation$b_m + validation$b_u))

  if (return_prediction){
    return(list("rmse"       = my_rmse,
                "prediction" = mu + validation$b_m + validation$b_u))
  }else{
    return(my_rmse)
  }
}

# calculate errors for a set of lambda values and choose the smallest rmse
lambdas <- seq(0, 10, 0.25)
model_rmses <- sapply(lambdas, Movie_User_Effect_Model)
lambda_of_smallest_rmse <- lambdas[which.min(model_rmses)]

# plot it
ggplot() + geom_line(aes(lambdas, model_rmses), col="#56B4E9", size=1) +
  geom_vline(xintercept=lambda_of_smallest_rmse, col="red") + ggtitle("Movie Effect Model: RMSE as func
```




```
# add the best prediction
results_temp <- tibble("Model" = "movie_user_effect",
                       "RMSE"  = model_rmses[which.min(model_rmses)])
results <- results %>% bind_rows(results_temp)
# show results table in latex format
kable(results_temp, "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

Model	RMSE
movie_user_effect	0.864817

Conclusion

The model that has the smallest RMSE is the Movie User Effect model

```
kable(results, "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

Model	RMSE
mean	1.0612018
median	1.1680160
mean_per_user	0.9783360
median_per_user	1.0211364
mean_per_movie	0.9439087
median_per_movie	0.9716910
mean_per_year	1.0500259
median_per_year	1.0661375
mean_per_genre	1.0702479
ols	0.8786586
movie_effect	0.9438521
movie_user_effect	0.8648170