MovieLens Project

Diógenes Carroz

15-08-2020

Netflix challenger

"The company's challenge, begun in October 2006, was both geeky and formidable: come up with a recommendation software that could do a better job accurately predicting the movies customers would like than Netflix's in-house software, Cinematch. To qualify for the prize, entries had to be at least 10 percent better than Cinematch."

1. Overview

These are the instructions given to develop this project:

"For this project, you will be creating a movie recommendation system using the MovieLens dataset.

The version of movielens included in the dslabs package (which was used for some of the exercises in PH125.8x: Data Science: Machine Learning) is just a small subset of a much larger dataset with millions of ratings. You can find the entire latest MovieLens dataset here. You will be creating your own recommendation system using all the tools we have shown you throughout the courses in this series. We will use the 10M version of the MovieLens dataset to make the computation a little easier

Develop your algorithm using the edx set. For a final test of your algorithm, predict movie ratings in the validation set as if they were unknown. RMSE will be used to evaluate how close your predictions are to the true values in the validation set.

Important: The validation data should NOT be used for training your algorithm and should ONLY be used for evaluating the RMSE of your final algorithm. You should split the edx data into separate training and test sets to design and test your algorithm."

Overall objective

To train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.

Specific objectives

- Develop your algorithm using the edx set
- Split the edx data into separate training and test sets
- Supply the following:
    1. A report in the form of an Rmd file.
    2. A report in the form of a PDF document knit from your Rmd file.
    3. A script in R format that generates your predicted movie ratings and RMSE score

Goal

Design an algorithm that provides predictions of ratings with an **$RMSE less than 0.86490**.

Create Train and Validation Sets

The following code to generate your datasets. This code was given as part of the project instructions:

```
################################
# Create edx set, validation set
################################

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

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(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)
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, sample.kind="Rounding")

# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
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)
```

Recommendation systems

Recommendation systems use given ratings to make specific recommendations. High ratings that serve to give recommendations to a certain user are predicted.

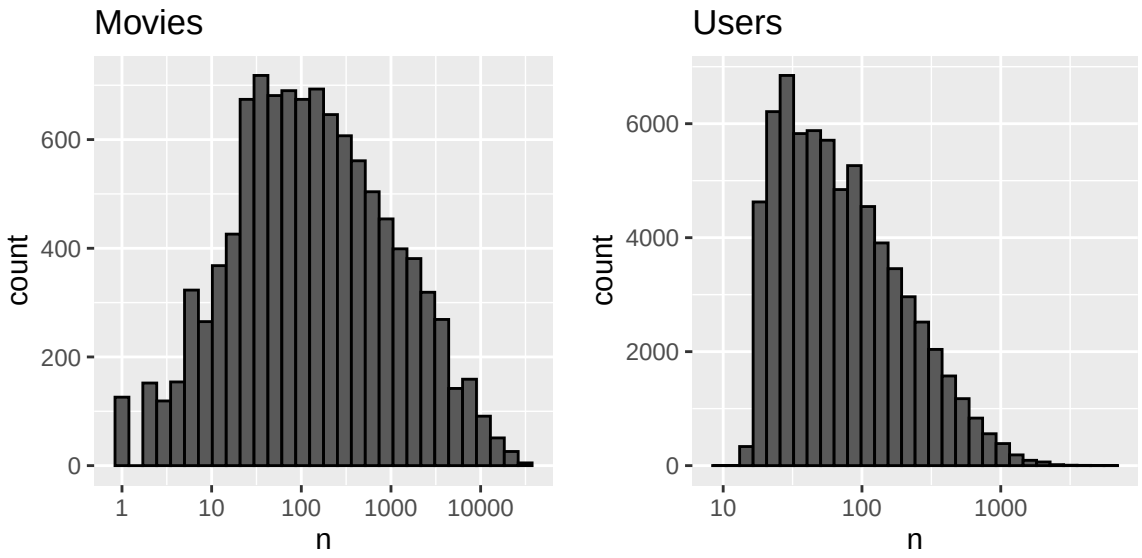
Netflix uses a star-based recommendation system that a user will give a movie. 1 star means not a good movie and 5 stars for excellent movie. Netflix offered a challenge to improve its recommendation algorithm by 10%. The winner earned an RMSE of 0.857.

Netflix data is not public. For this project use half of the GroupLens data (10 million ratings) on movie ratings.

2. Analysis

We will call each result of the prediction algorithm Y. We use different predictors: for the movie: i, for the user: u, and others that will be added later.

Some movies have higher ratings than others and some users are more active in rating:



Loss function

The effectiveness of the algorithm will be evaluated based on the residual mean square error (RMSE) in a test set. Defining $Y_{u,i}$ as the rating of the movie i by the user u and the prediction with $\hat{Y}_{u,i}$. The RMSE is then defined as:

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

In this way we obtain the `RMSE` function in R code:

```
#############################################################################
## RMSE function
##
## Calculate RMSE (residual mean squared error)
#############################################################################
RMSE <- function(true_ratings, predicted_ratings){
    sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Create Train and Test set

Starting from the data given in the edx dataframe we will create a set of train and a set of tests to evaluate the precision of the algorithm.

```
#############################################################################
## create_sets function
##
## Create Train set and test set
```

```
#############################################################################
create_sets <- function(data) {

    # date represents the duration in weeks of the rating record in the data
    data <- mutate(data, date = round_date(as_datetime(timestamp), unit = "week"))
    data <- data %>% select(rating, userId, movieId, date, genres)

    # test set will be 10% of data
    set.seed(1, sample.kind="Rounding")
    # if using R 3.5 or earlier, use `set.seed(1)` instead
    test_index <- createDataPartition(y = data$rating, times=1, p=0.1, list=FALSE)
    train_set <- data[-test_index,]
    temp <- data[test_index,]

    # Make sure userId, movieId, date and genres in test set are also in train set
    test_set <- temp %>%
        semi_join(train_set, by = "movieId") %>%
        semi_join(train_set, by = "userId") %>%
        semi_join(train_set, by = "date") %>%
        semi_join(train_set, by = "genres")

    # Add rows removed from test set back into train set
    removed <- anti_join(temp, test_set)
    train_set <- rbind(train_set, removed)

    return(list(train = train_set,test = test_set))
}
```

```
## Separate edx into train and test set
sets <- create_sets(edx)
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
## Joining, by = c("rating", "userId", "movieId", "date", "genres")
```

```
train_set <- sets$train
test_set <- sets$test
rm(edx, sets)
```

Table 1: First rows of train_set

	rating	userId	movieId	date	genres
1	5	1	122	1996-08-04	Comedy|Romance
3	5	1	292	1996-08-04	Action|Drama|Sci-Fi|Thriller
4	5	1	316	1996-08-04	Action|Adventure|Sci-Fi
5	5	1	329	1996-08-04	Action|Adventure|Drama|Sci-Fi
6	5	1	355	1996-08-04	Children|Comedy|Fantasy
7	5	1	356	1996-08-04	Comedy|Drama|Romance|War

Building the algorithm

A first model: average

Let's start by giving a rating to the movies independent of the user, for this we will use the average, μ, of the movie ratings given by all users, and $\epsilon_{u,i}$ independent errors sampled from the same distribution centered at 0:

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

So we calculate the average `train_set$rating` and store it in `mu_hat`:

```
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512456
```

Thus, we obtain the first predictions and calculate its RMSE as follows:

```
rmse <- RMSE(test_set$rating, mu_hat)
rmse
```

```
## [1] 1.060054
```

The RMSE is approximately 1, which is far from the goal of 0.8649 and even more than the 0.857 of the challenge winner. So we must continue to improve the model.

Second Model: movie effect

Since movies are rated differently we can add the term b_i ("bias") to the model to represent the average rating for movie i:

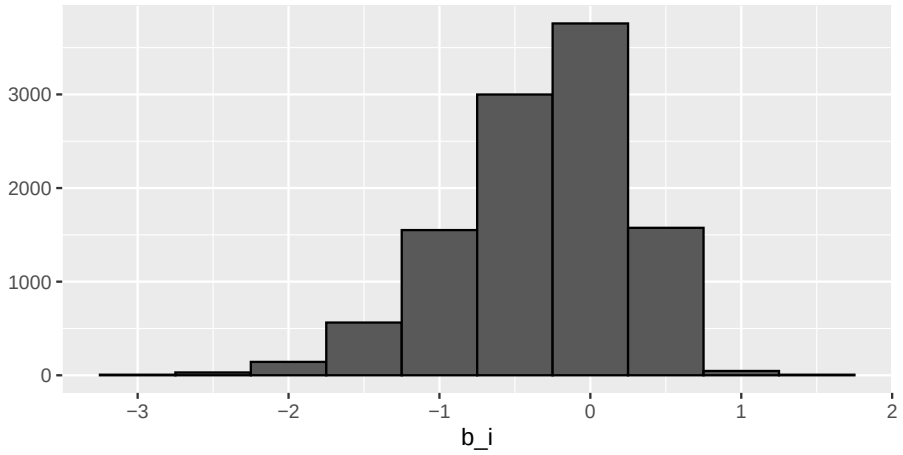
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

If we use the `lm` function to estimate the values of b_i, for example: `fit <- lm(rating ~ as.factor(movieId), data = train_set)`, it will be very slow due to the size of the data. So it is recommended to use the least squares estimate $\hat{b}_i$ is just the average of $Y_{u,i} - \hat{\mu}$ for each movie i, where the hat in the code to represent the estimates in the future:

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

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

We can see that these estimates vary substantially:

```
qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```



Let's calculate the predictions with the test set:

```
predicted_ratings <- mu + test_set %>%
    left_join(movie_avgs, by='movieId') %>%
    pull(b_i)

rmse <- RMSE(predicted_ratings, test_set$rating)
rmse
```

```
## [1] 0.9429615
```

Table 2: Results up to model 2

method	RMSE
Model 1: Average	1.0600537
Model 2: Average + Movie	0.9429615

We have not reached the goal yet so we will continue adding bias

Third Model: user effect

Now we add the effect that the user has on the ratings

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

where b_u is a user-specific effect. The negative or positive effects that could be caused by the mood of the users at the time of rating the film are counteracted and obtain a better prediction.

We can estimate $\hat{b}_u$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i$, and express it with the following code:

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

We calculate the predictors and the RMSE:

```
predicted_ratings <- test_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
```

```
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

rmse <- RMSE(predicted_ratings, test_set$rating)
rmse
```

```
## [1] 0.8646843
```

Table 3: Results up to model 3

method	RMSE
Model 1: Average	1.0600537
Model 2: Average + Movie	0.9429615
Model 3: Average + Movie and User effect	0.8646843

There is an improvement but it is not enough yet.

Fourth Model: Regularization

The variability of the number of movie ratings (movies with few ratings versus movies with many ratings) introduces a distorting effect, so regularization tries to avoid this effect.

The general idea is to control the total variability of the movie effects, considering that instead of minimizing the least squares equation, minimizing an equation that adds a penalty:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i\right)^2+\lambda\sum_i b_i^2$$

"The first term is for least squares only and the second is a penalty that increases when many b_i are large. Using calculus we can show that the values of b_i that minimize this equation are:"

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

When the sample size n_i is very large, the estimate will be stable, and the penalty λ. However, when n_i is small, then the estimate $\hat{b}_i\left(\lambda\right)$ is reduced to 0.

Note: if at this point you are a bit confused as I was when I saw this explanation for the first time, I clarify that the λ is just a tuning parameter, what a friend accountant would say a PQC (in Spanish: "Para que cuadre") or "so that it fits".

I know that a lambda close to 5 gives better results, however I will show you the results of the model with the tuning that they taught us:

If we continue with the explanation given about the regularization we will see that it is applied only with the effect of the movies, it does not improve the RMSE, because I will skip this explanation and show the results with the effect of the regularization when the effect of the users is added.

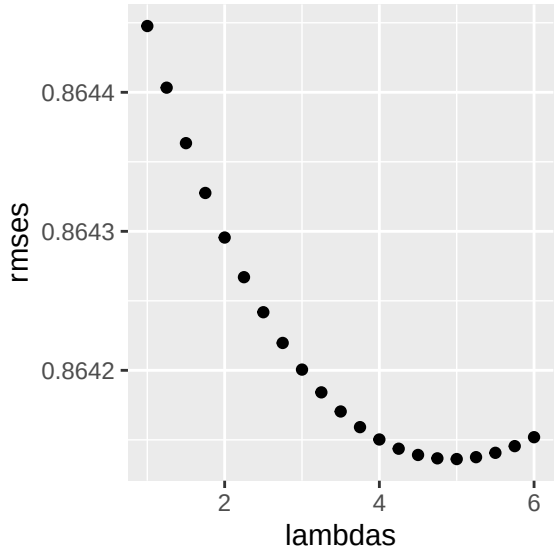
Just to show how unnecessarily confusing the explanation they gave us, I show you the following formula:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_i b_u^2\right)$$

It is the same as the previous one but the regularization of the user effect is added.

With this code we have the λ that introduces the greatest improvement to our algorithm. Note: just shorten it to 6 to "make it different":

```
lambdas <- seq(1, 6, 0.25)

rmses <- sapply(lambdas, function(l){
    mu <- mean(train_set$rating)

    b_i <- train_set %>%
        group_by(movieId) %>%
        summarize(b_i = sum(rating - mu)/(n()+l))

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

    predicted_ratings <-
        test_set %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        mutate(pred = mu + b_i + b_u) %>%
        pull(pred)

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



For the full model, the optimal λ is:

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

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

And minimal RMSE is:

```
rmse <- min(rmses)
rmse
```

```
## [1] 0.8641362
```

Table 4: Results up to model 4

method	RMSE
Model 1: Average	1.0600537
Model 2: Average + Movie	0.9429615
Model 3: Average + Movie and User effect	0.8646843
Model 4: Average + Regularized Movie and User Effect Model	0.8641362

If you have read everything so far you will notice that I have basically explained the same thing that was given to us in the Marchine Learning course. I hope I do not get comments that I am only copy-pasting the course material, as I got in a previous evaluation from a person who surely did not read the full report.

From now on I will add other biases that were not in the course model. However, I avoid reinventing the wheel, and also there is no reason to win the million dollars so the bias will be predictable: the effect of how much has happened since the rating was given and the genre of the movies.

Fifth Model: date effect

The movielens dataset also includes a time stamp. This variable represents the time and data in which the rating was provided. The units are seconds since January 1, 1970.

There is some evidence of a time effect on average rating. We define $d_{u,i}$ as the day for user's u rating of movie i and we will use the smoothing technique to have the following model:

$$Y_{u,i} = \mu + b_i + b_u + f\left(d_{u,i}\right) + \epsilon_{u,i}$$

with f a smooth function of $d_{u,i}$

The general idea of smoothing is to group data points into strata in which the value of $f(x)$ can be assumed to be constant. We can make this assumption because we think $f(x)$ changes slowly and, as a result, $f(x)$ is almost constant in small windows of time.

Let's assume that users kept their opinion of the movies roughly the same in a week. With this assumption in place, we have multiple data points with the same expected value.

If we fix a day to be in the center of our week, call it x_0, then for any other day x such that $|x - x_0| \leq 3.5$, we assume $f(x)$ is a constant $f(x) = \mu$. This assumption implies that:

$$E\left[Y_i | X_i = x_i\right] \approx \mu \text{ if } |x - x_0| \leq 3.5$$

This assumption implies that a good estimate for $f(x)$ is the average of the Y_i values in the window. If we define A_0 as the set of indexes i such that $|x_i - x_0| \leq 3.5$ and N_0 as the number of indexes A_0 then our estimate is:

$$\hat{f}\left(x_0\right)) = \frac{1}{N_0} \sum_{i \epsilon A_0} Y_i$$

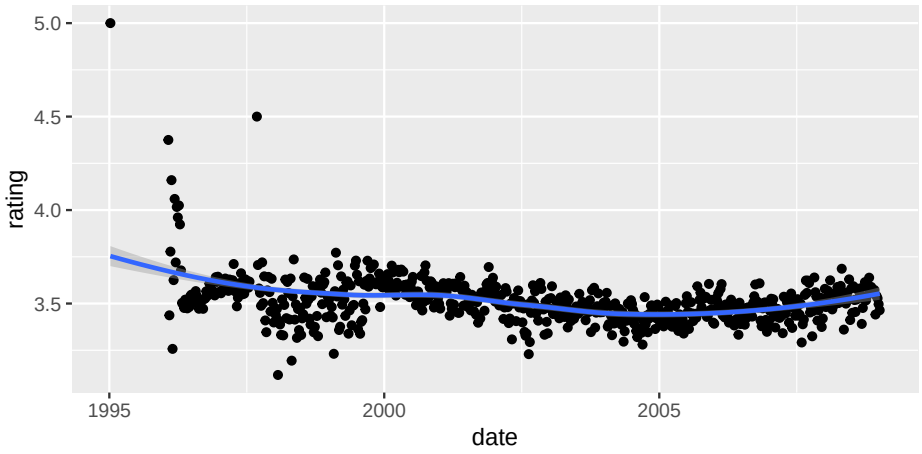
In the case of Movielens, we define the date column from timestamp and smooth it with a 7-day window (week):

```
train_set <- mutate(train_set, date = round_date(timestamp, unit = "week"))
```

The average rating for each week and plot this average against date:

```
train_set %>% group_by(date) %>%
    summarize(rating = mean(rating)) %>%
    ggplot(aes(date, rating)) +
    geom_point() +
    geom_smooth()
```

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



Here we create the function to calculate `predicter_ratings` the predictions of the ratings:

```
################################################################################
## predicter_ratings function
##
## Make Predicted Ratings for Movielens
################################################################################
predicter_ratings <- function(train_data, test_data, lambda) {
    ## Y_(u,i) = mu + b_i + b_u + f(d_u,i) + g_(u,i)

    # ratings' average
    mu <- mean(train_data$rating)

    ## Regularized Movie Effect
    b_i <- train_data %>%
        group_by(movieId) %>%
        summarize(b_i = sum(rating - mu)/(n() + lambda))

    ## Regularized User Effect
    b_u <- train_data %>%
        left_join(b_i, by="movieId") %>%
        group_by(userId) %>%
        summarize(b_u = sum(rating - b_i - mu)/(n() + lambda))

    ## Regularized timestamp effect
    b_d <- train_data %>%
        left_join(b_i, by="movieId") %>%
        left_join(b_u, by="userId") %>%
```

```
        group_by(date) %>%
        summarize(b_d = sum(rating - b_u - b_i - mu)/(n() + lambda))

    ## Predicted Ratings with Regularized Movie + User + date Effect Model
    predicted_ratings <- test_data %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_d, by = "date") %>%
        mutate(pred = mu + b_i + b_u + b_d) %>%
        .$pred

    return(predicted_ratings)
}
```

We also create a function `training_lambda` that calculates the lambda:

```
##########################################################################
## training_lambda function
##
## Calculate the best lambda for prediction
##########################################################################
training_lambda <- function(train_data, test_data, lbmin=1, lbmax=10, s=1, ite=1) {
    # Perform "ite" iterations to calculate the best lambda between
    # the values "lbmin" and "lbmax"

    s <- max(s,1)
    ite <- max(ite,1)
    br <- 99999
    bl <- 0
    lambdas <- seq(lbmin, lbmax, s)
    for(i in 1:ite) {

        message("   Iteration ", i)

        rmses <- sapply(lambdas, function(l) {
            r <- RMSE(predicter_ratings(train_data, test_data, l) ,test_data$rating)
            message("      RMSE = ", round(r,6), "  Lambda = ",l)
            r
        })

        minr <- min(rmses)
        lminr <- lambdas[which.min(rmses)]

        # Determine the best lambda
        if(minr < br) {
            br <- minr
            bl <- lminr
        }
        message("   ----------------------------------------")
        message("   Minimum RMSE = ", round(br,6), "  Lambda = ",bl)
        message("   ----------------------------------------")

        # Cut the gap in half and fix the range around the best lambda
        s <- s/2
```

```
        lambdas <- c(bl - s, bl + s)
    }
    return(bl)
}
```

We build a new model by adding the smoothing effect of the timestamp, expressed by date:

```
## Calculate the best lambda
lambda <- training_lambda(train_set, test_set, 4, 6, 1, 3)
```

```
##    Iteration 1
##       RMSE = 0.86404  Lambda = 4
##       RMSE = 0.864018  Lambda = 5
##       RMSE = 0.864026  Lambda = 6
##    ----------------------------------------
##    Minimum RMSE = 0.864018  Lambda = 5
##    ----------------------------------------
##    Iteration 2
##       RMSE = 0.864025  Lambda = 4.5
##       RMSE = 0.864019  Lambda = 5.5
##    ----------------------------------------
##    Minimum RMSE = 0.864018  Lambda = 5
##    ----------------------------------------
##    Iteration 3
##       RMSE = 0.864021  Lambda = 4.75
##       RMSE = 0.864018  Lambda = 5.25
##    ----------------------------------------
##    Minimum RMSE = 0.864018  Lambda = 5.25
##    ----------------------------------------
```

```
## Calculate ratings
predicted_ratings <- predicter_ratings(train_set, test_set, lambda)

rmse <- RMSE(predicted_ratings, test_set$rating)
rmse
```

```
## [1] 0.8640175
```

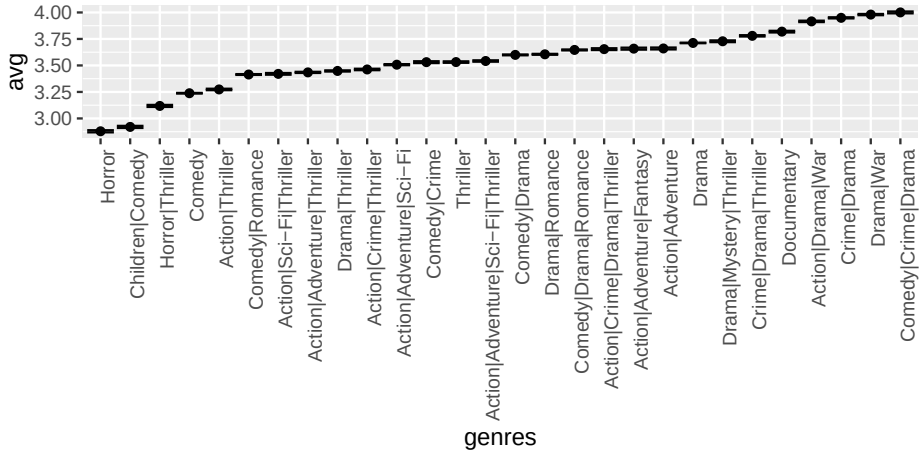
Table 5: Results up to model 5

method	RMSE
Model 1: Average	1.0600537
Model 2: Average + Movie	0.9429615
Model 3: Average + Movie and User effect	0.8646843
Model 4: Average + Regularized Movie and User Effect Model	0.8641362
Model 5: Average + Regularized Movie, user and date Effect Model	0.8640175

Sixth Model: Genres effect

The movielens data also has a `genres` column. This column includes every genre that applies to the movie.

```
train_set %>% group_by(genres) %>%
    summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
    filter(n >= 50000) %>%
    mutate(genres = reorder(genres, avg)) %>%
    ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
    geom_point() +
    geom_errorbar() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



There are genres of movies that receive higher ratings than other genres. If we define $g_{u,i}$ as the genre for user u's rating of movie i, we have the following model:

$$Y_{u,i} = \mu + b_i + b_u + f\left(d_{u,i}\right) + g_{u,i} + \epsilon_{u,i}$$

3. Results

Finally, a model is presented with an algorithm that provides rating predictions with an RMSE that satisfies the goal of the project. Although you do not reach an RMSE of 0.857, nor do you earn the $ 1,000,000; I hope it is enough to get a good grade.

We only need to update the `predicter_ratings` function to add the genres effect:

```
predicter_ratings <- function(train_data, test_data, lambda) {
    ## Y_(u,i) = mu + b_i + b_u + f(d_u,i) + g_(u,i)

    # ratings' average
    mu <- mean(train_data$rating)

    # Regularized Movie Effect
    b_i <- train_data %>%
        group_by(movieId) %>%
        summarize(b_i = sum(rating - mu)/(n() + lambda))

    # Regularized User Effect
    b_u <- train_data %>%
        left_join(b_i, by="movieId") %>%
```

```
        group_by(userId) %>%
        summarize(b_u = sum(rating - b_i - mu)/(n() + lambda))

    # Regularized date (timestamp) effect
    b_d <- train_data %>%
        left_join(b_i, by="movieId") %>%
        left_join(b_u, by="userId") %>%
        group_by(date) %>%
        summarize(b_d = sum(rating - b_u - b_i - mu)/(n() + lambda))

    # Regularized genres effect
    b_g <- train_data %>%
        left_join(b_i, by="movieId") %>%
        left_join(b_u, by="userId") %>%
        left_join(b_d, by="date") %>%
        group_by(genres) %>%
        summarize(b_g = sum(rating - b_d - b_u - b_i - mu)/(n() + lambda))

    # Predicted Ratings with Regularized Movie + User + date + genres Effect Model
    predicted_ratings <- test_data %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_d, by = "date") %>%
        left_join(b_g, by = "genres") %>%
        mutate(pred = mu + b_i + b_u + b_d + b_g) %>%
        .$pred

    return(predicted_ratings)
}
```

We calculate the predictions of the ratings with our train set and validate it with the test set:

```
## Calculate ratings
predicted_ratings <- predicter_ratings(train_set, test_set, lambda)

rmse <- RMSE(predicted_ratings, test_set$rating)
rmse
```

```
## [1] 0.8636737
```

Table 6: Results up to model 6

method	RMSE
Model 1: Average	1.0600537
Model 2: Average + Movie	0.9429615
Model 3: Average + Movie and User effect	0.8646843
Model 4: Average + Regularized Movie and User Effect Model	0.8641362
Model 5: Average + Regularized Movie, user and date Effect Model	0.8640175
Model 6: Average + Regularized Movie, User, Date and Genres Effect Model	0.8636737

Finally, we calculate our predictions with **validation set** and compare our results. We make sure there are dates and genres are present in the validation set.

```
# Make sure date and genres in validation set are also in train set
temp <- mutate(validation, date = round_date(as_datetime(timestamp), unit = "week"))
validation <- temp %>%
    semi_join(train_set, by = "date") %>%
    semi_join(train_set, by = "genres")
    removed <- anti_join(temp, validation)
```

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

```
train_set <- rbind(train_set, removed)
rm(temp, removed)

## Calculate predictions
predicted_ratings <- predicter_ratings(train_set, validation, lambda)

## Calculate RMSE
rmse <- RMSE(predicted_ratings, validation$rating)
rmse
```

```
## [1] 0.864703
```

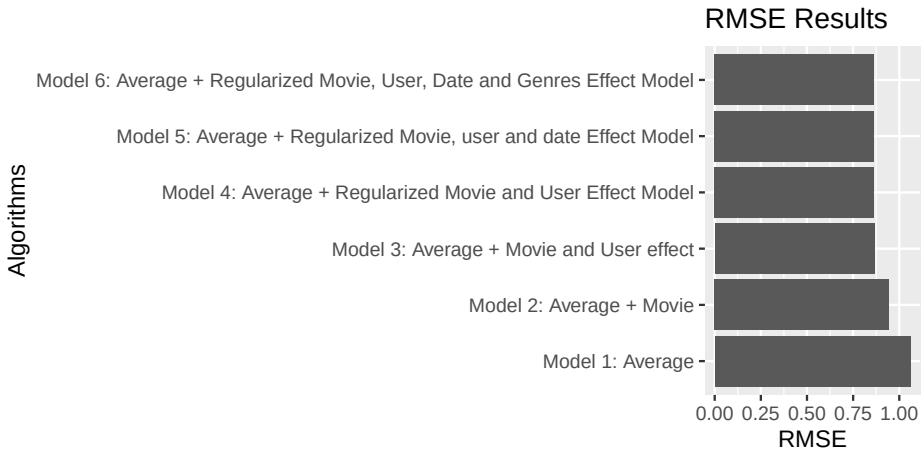
4. Conclusion

In the process of obtaining an algorithm with a low RMSE, below the set goal: **0.86490**, variables were added to the model, starting with the average and adding the Bias associated with the movie, user, rating date and the gender of the movie.

We express such a progression in equations by adding the effect as follows:

- Average: $Y_{u,i} = \mu + \epsilon_{u,i}$
- Movie: $Y_{u,i} = \mu + b_i + \epsilon_{u,i}$
- User: $Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$
- Date: $Y_{u,i} = \mu + b_i + b_u + f\left(d_{u,i}\right) + \epsilon_{u,i}$
- Genre: $Y_{u,i} = \mu + b_i + b_u + f\left(d_{u,i}\right) + g_{u,i} + \epsilon_{u,i}$

```
rmse_results %>%
ggplot(aes(reorder(method,-RMSE), x = RMSE)) +
    geom_col() +
    ggtitle("RMSE Results") +
    labs(y = "Algorithms")
```



The regularization improves the algorithm attached to determining the best lambda.

Finally, the resulting model to predict ratings with the validation set with an RMSE, obtaining a better value than the target.

RMSE:

```
## [1] 0.864703
```

I apologize for some spelling and semantic errors. As I am not fluent in English, he asked a friend (the translator of the browser that begins with G) to do the translations for me.

And even if you think that I have taken some paragraphs and formulas of the courses and I have only rephrased it, I assure you that I have studied and practiced all of them. This is the second version of my report, I was able to change everything and improve the model with what I saw in the evaluations I did. But I decided to only make corrections in the wording and in some formulas. I must admit that I had a lot more fun writing the report than doing the script. I had to learn more about the .Rmd documents in order to reproduce the formulas, center the graphics, and change the font styles.

I thank you if you have read the entire report and not only gave it a quick look like others did.