HarvardX : Capstone, Movielens Project

Edward Ryan

Introduction

The goal of a movie recommendation system is to foresee how a given user will rate a particular movie dependent on the user and the movie itself. Therefore, this problem allows numerous creative ways to tackle it. This particular project uses the Movielens dataset. The full MovieLens dataset, which can be found here: https://grouplens.org/datasets/movielens/latest/, is rather large. To make the computation easier, this project uses the "10M" version of Movielens dataset instead, which can be found here: https://grouplens.org/datasets/movielens/10m/. This dataset has around 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. The basis for evaluation in this project is through calculating the RMSE (Residual Mean Squared Error) from the predicted ratings and the actual ratings. From the final model, it can be inferred that the best model implements user, movie, timestamp, and genre effects with regularization on large values. Furthermore, it incorporates matrix factorization from the residuals which greatly improves the result.

Methods

Installing packages

First, we must set up and load the necessary packages for the code to run.

```
install.packages("tidyverse")
install.packages("lubridate")
install.packages("caret")
install.packages("recosystem")
install.packages("data.table")
install.packages("gridExtra")
```

```
library(tidyverse)
library(lubridate)
library(caret)
library(recosystem)
library(data.table)
library(gridExtra)
```

Downloading dataset

As mentioned in the introduction, this project uses the "10M" version of the Movielens dataset. Information regarding the dataset can be found here: https://grouplens.org/datasets/movielens/10m/. The dataset can be downloaded here: http://files.grouplens.org/datasets/movielens/ml-10m.zip. The following code will download the dataset and combine ratings and movies into *movielens*.

```
# Creates a temporary directory to store the downloaded data
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
```

```
# Collecting the ratings and movies from the data
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))

# Combining ratings and movies into movielens
movielens <- left_join(ratings, movies, by = "movieId")
```

Furthermore, we will split *movielens* into train data (*edx*) and validation data (*validation*). The validation data will be 10% of *movielens* data. We must make sure that *validation* contains the same userIds and movieIds as in *edx*.

```
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead

# Validation will be 10% of movielens
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)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

We must always remember that the ratings in *validation* should not be used in the process of getting the predicted ratings.

Evaluation for results

We will define a function that calculates RMSE (Residual Mean Squared Error) that will evaluate the performance of our models later on. It will calculate the difference between our predicted ratings and the actual ratings.

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

We will also create a table in *rmse_results* to store our models' RMSE/performance later on.

```
rmse_results <- tibble()
```

Data Exploration & Modelling Strategy

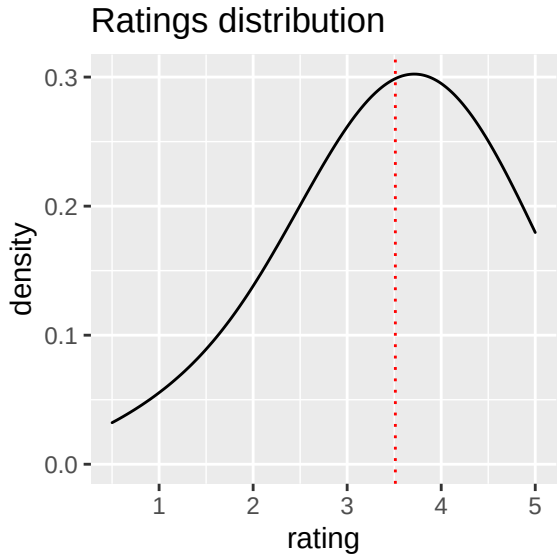
Baseline models

First, we must know the data that we are dealing with. Let us see the ratings' distribution. The distribution is left-skewed.

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

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

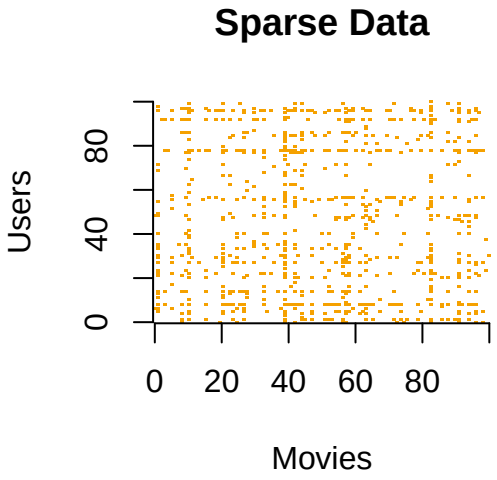
```
edx %>%
  ggplot(aes(rating)) +
  geom_density(adjust = 30) +
  geom_vline(xintercept = mean(edx$rating), linetype = "dotted", color = "red") +
  ggtitle("Ratings distribution")
```



Our *edx* dataset has 69878 users and 10677 movies. If every user rated every movie available, we would have 746087406 ratings. However, our dataset only has 9000055 ratings. This number is only 1.2063% of all possible user movie ratings. This fact can be seen more clearly with a visualization. The plot below is derived from 100 random samples of users and 100 random samples of movies. We can think of this recommendation challenge as filling in the blank spots with our predicted ratings. We want to recommend the right movies for every person using our available data.

```
temp <- edx %>%
  filter(userId %in% sample(edx$userId, 100, replace = FALSE)) %>%
  select(userId, movieId, rating) %>%
  spread(key = movieId, value = rating)
ncol <- ncol(temp)
temp <- temp %>% select(sample(1:ncol, 100, replace = FALSE)) %>% as.matrix()
temp[!is.na(temp)] <- 1
```

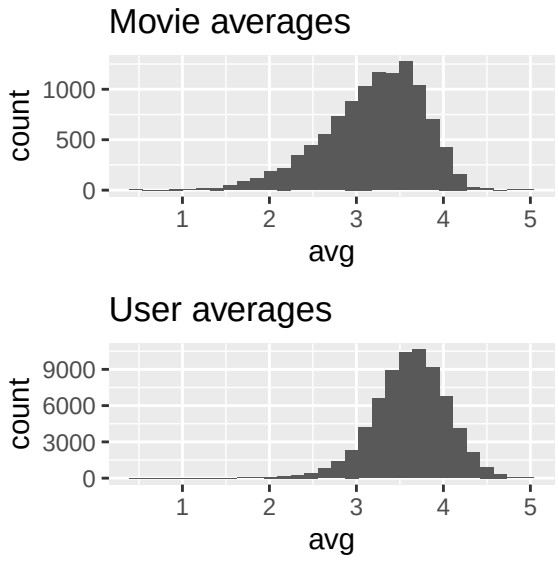
```
temp %>% image(xlab = "Movies", ylab = "Users", axes = FALSE, main = "Sparse Data")
axis(1, at = seq(0.0,1.0,0.2), labels = seq(0,100,20))
axis(2, at = seq(0.0,1.0,0.2), labels = seq(0,100,20))
```




```
rm(temp)
```

Logically speaking, every movie and user will have different averages upon their ratings. Some movies are objectively bad or good. Some users are very generous in giving ratings and some are not. The plot below shows that this statement is proven with data. We will implement movie and user averages into our model.

```
# Different movie and user averages
plot_1 <- edx %>%
  group_by(movieId) %>%
  summarize(avg = mean(rating)) %>%
  ggplot(aes(avg)) +
  geom_histogram() +
  ggtitle("Movie averages")

plot_2 <- edx %>%
  group_by(userId) %>%
  summarize(avg = mean(rating)) %>%
  ggplot(aes(avg)) +
  geom_histogram() +
  ggtitle("User averages")

grid.arrange(plot_1, plot_2)
```

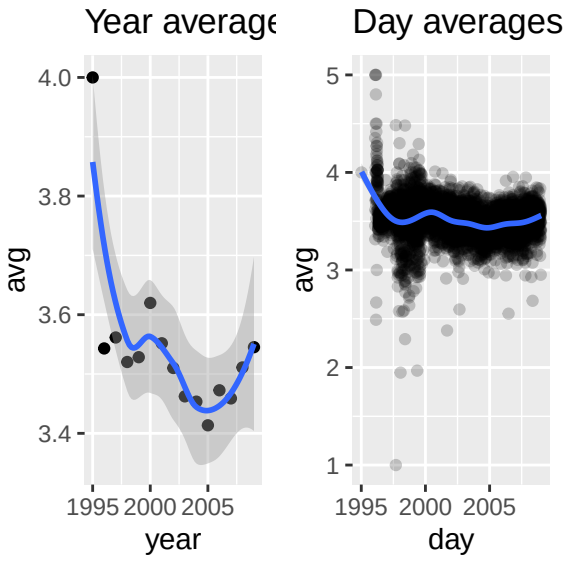



```
rm(plot_1, plot_2)
```

From *edx* data, we have the *timestamp* variable. It turns out that the averages for timestamp rounded to the nearest day and year are different. The graph below will show how the averages change over some time. The year averages graph shows the changes more clearly. However, it turns out that rounding the timestamp to the nearest day will provide the best RMSE result among any other time metric.

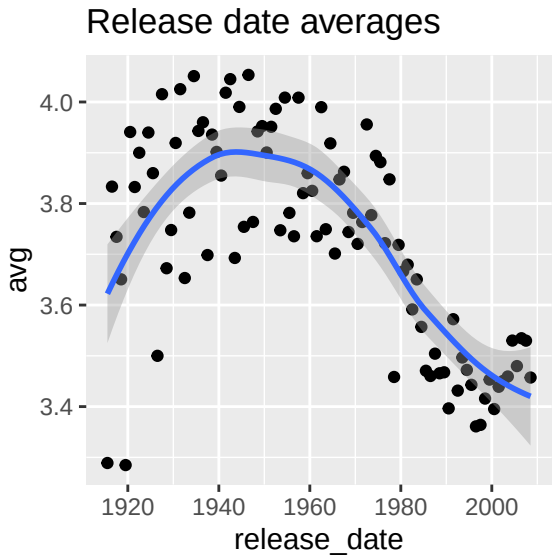
```
# Mutate edx and validation
edx <- edx %>%
  mutate(day = timestamp %>% as_datetime() %>% round_date(unit = "day"),
         year = timestamp %>% as_datetime() %>% round_date(unit = "year"))
validation <- validation %>%
  mutate(day = timestamp %>% as_datetime() %>% round_date(unit = "day"),
         year = timestamp %>% as_datetime() %>% round_date(unit = "year"))
```

```
# Year averages
plot_1 <- edx %>%
  group_by(year) %>%
  summarize(avg = mean(rating)) %>%
  ggplot(aes(year, avg)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Year averages")

# Day averages
plot_2 <- edx %>%
  group_by(day) %>%
  summarize(avg = mean(rating)) %>%
  ggplot(aes(day, avg)) +
  geom_point(alpha = 0.2) +
  geom_smooth() +
  ggtitle("Day averages")

grid.arrange(plot_1, plot_2, ncol = 2)
```



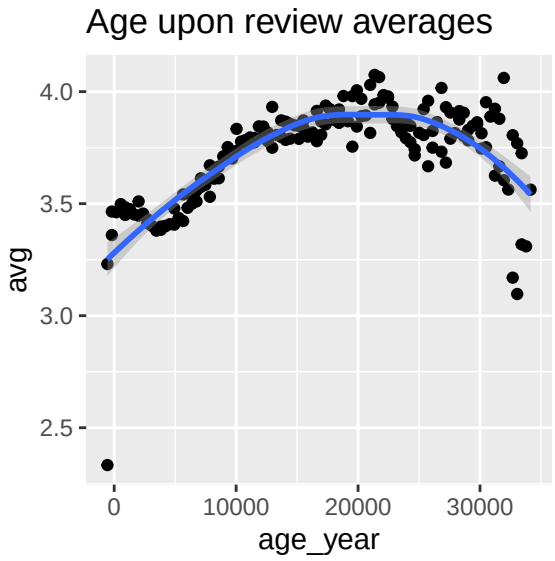
If we extract the release date of the movies from the *title* variable, we will gain more valuable data that can be integrated into our model. These release dates will also have different averages. The graph below shows that in the beginning, rating averages are getting higher. A possible reason for this is that viewers become more acquainted with movies as it is still a newly developed entertainment and media. However, the average reaches its peak between 1940 and 1960 when after that, the ratings plummeted to a low average. A possible reason to explain this is that the supply of movies is so great and that people now can differentiate what a good or bad movie is. Despite the numerous interpretations of the graph below, we must still implement this predictor into our model.

```
# Release date averages
# Extracting release date
release_date <- edx$title %>%
  str_extract(pattern = "\\(\\d{4}\\)$") %>%
  str_replace(pattern = "\\(", replacement = "") %>%
  str_replace(pattern = "\\)", replacement = "")
edx <- edx %>%
  mutate(release_date = release_date)
release_date <- validation$title %>%
  str_extract(pattern = "\\(\\d{4}\\)$") %>%
  str_replace(pattern = "\\(", replacement = "") %>%
  str_replace(pattern = "\\)", replacement = "")
validation <- validation %>%
  mutate(release_date = release_date)
rm(release_date)

edx %>%
  mutate(release_date = as_date(release_date, format = "%Y")) %>%
  group_by(release_date) %>%
  summarize(avg = mean(rating)) %>%
  ggplot(aes(release_date, avg)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Release date averages")
```



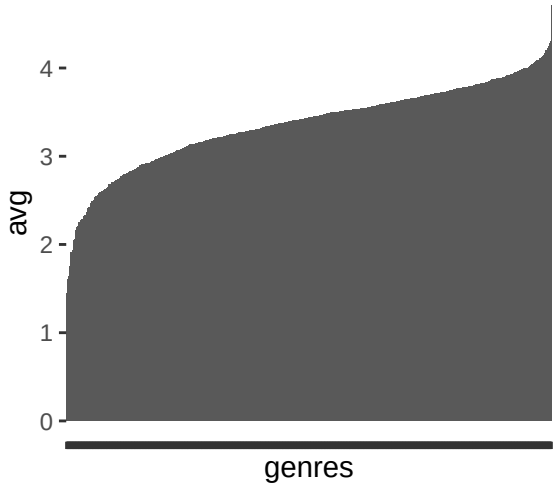
Furthermore, if we subtract the timestamp with the release date, we will get the age upon rating. This data is also valuable and will be implemented in our model later. The graph below shows that in the beginning, as time goes by, the averages get higher. This truth can lead to the fact that as more people rate a movie, the higher the average is. Or, another way to explain it is that movies have gone into the selection stage, where users will rate an old movie because it is recommended by others. However, the average has its peak and soon plummeted. Maybe, when a movie gets too old, its style and character do not fit anymore with most users. Nevertheless, these statements are just all interpretations.

```
# Age upon review averages
# Adding age by year
age <- as.Date(edx$year) - as.Date(edx$release_date, format = "%Y")
edx <- edx %>%
  mutate(age_year = age)
age <- as.Date(validation$year) - as.Date(validation$release_date, format = "%Y")
validation <- validation %>%
  mutate(age_year = age)
rm(age)

edx %>%
  group_by(age_year) %>%
  summarize(avg = mean(rating)) %>%
  ggplot(aes(age_year, avg)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Age upon review averages")
```



Lastly, we will implement genres averages into our model. The *genres* variable consists of combinations of genres a particular movie has. We can separate the genres and find averages for every genre. However, hypothetically speaking, a combination of genres will not simply be expressed by adding their averages. Some combinations of genres can have a negating effect. In other words, some genres just do not go well together. This information will not be explained if we add averages for different genres. Therefore, it is wise for us to take averages for combinations of genres directly. It turns out that doing so will provide a better RMSE.

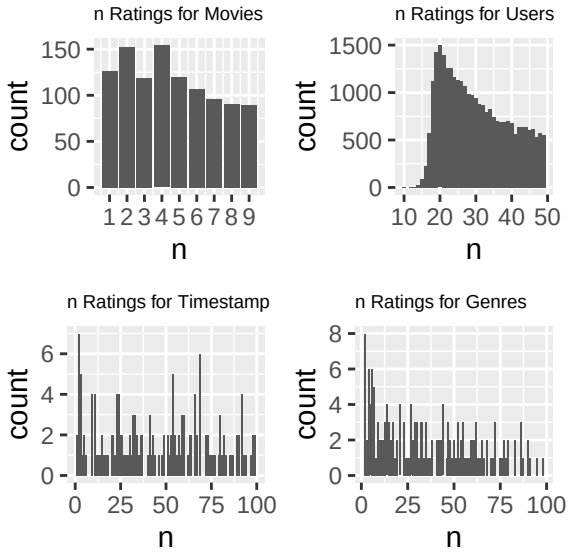
```
# Genre averages
edx %>%
  group_by(genres) %>%
  summarize(avg = mean(rating)) %>%
  arrange(avg) %>%
  ggplot(aes(x = reorder(genres, avg), y = avg)) +
  geom_col() +
  theme(axis.text.x = element_blank()) +
  xlab("genres") +
  ggtitle("Genre averages")
```



Regularization

The graph below shows that there are numerous cases where the number of ratings is low. This truth will mean that their averages will be less reliable as their true average needs a huge number of ratings. To combat this, we will use regularization later on to reduce very large or very small averages.

```
# n Ratings for Movies
plot_1 <- edx %>%
  group_by(movieId) %>%
  summarize(n = n()) %>%
  filter(n < 10) %>%
  ggplot(aes(n)) +
  geom_bar() +
  ggtitle("n Ratings for Movies") +
  theme(plot.title = element_text(size = 7)) +
  scale_x_continuous(breaks = seq(1:9))

# n Ratings for Users
plot_2 <- edx %>%
  group_by(userId) %>%
  summarize(n = n()) %>%
  filter(n < 50) %>%
  ggplot(aes(n)) +
  geom_bar() +
  ggtitle("n Ratings for Users") +
  theme(plot.title = element_text(size = 7)) +
  scale_x_continuous()

# n Ratings for Timestamp (Day)
plot_3 <- edx %>%
  group_by(day) %>%
  summarize(n = n()) %>%
  filter(n < 100) %>%
  ggplot(aes(n)) +
  geom_bar() +
  ggtitle("n Ratings for Timestamp") +
  theme(plot.title = element_text(size = 7)) +
  scale_x_continuous()

# n Ratings for Genres
plot_4 <- edx %>%
  group_by(genres) %>%
  summarize(n = n()) %>%
  filter(n < 100) %>%
  ggplot(aes(n)) +
  geom_bar() +
  ggtitle("n Ratings for Genres") +
  theme(plot.title = element_text(size = 7)) +
  scale_x_continuous()

# Combining plots
grid.arrange(plot_1, plot_2, plot_3, plot_4, ncol = 2, nrow = 2)
```




```
rm(plot_1, plot_2, plot_3, plot_4)
```

```
# n Ratings for Release Date
edx %>%
  group_by(release_date) %>%
  summarize(n = n()) %>%
  arrange(n) %>%
  head() %>%
  knitr::kable()
```

release_date	n
1917	32
1918	73
1916	84
1919	158
1915	180
1923	316

```
# n Ratings for Age (Year)
edx %>%
  group_by(age_year) %>%
  summarize(n = n()) %>%
  arrange(n) %>%
  head(n = 15) %>%
  knitr::kable()
```

age_year	n
-552 days	6
34147 days	8
33415 days	11
-553 days	13
33050 days	13
32685 days	18
33416 days	20
33781 days	21

age_year	n
33051 days	31
31955 days	43
31225 days	48
32686 days	53
31954 days	90
31590 days	93
31589 days	128

Matrix factorization

What we have done so far is implementing basic predictors into our model. These predictors explain information relating to each individual movies, users, and others. However, to integrate information that explains correlation between groups of movies and groups of users, we need to use matrix factorization. Turns out, this will improve our RMSE significantly.

Modeling strategy conclusion

Our model will include baseline predictors, which are movies + users + timestamp + release date + age + genres effects. Using these baseline models will produce an acceptable RMSE. However, it turns out that there are some improvements than we can make to improve the RMSE. The data shows that some averages have a few numbers of ratings; therefore, regularization can be implemented. We will optimize the lambda in regularization with cross-validation (folds). Lastly, we will incorporate matrix factorization that will explain information and correlation about groups of movies and groups of users with each other.

Results

Model 1 (Just the average)

Our first model will predict the average ratings. We predict the average ratings to minimize RMSE. It turns out our RMSE is 1.0612018.

```
mu <- mean(edx$rating)
prediction <- rep(mu, nrow(validation))
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = "Just the average",
                                 RMSE = RMSE(validation$rating, prediction)))
rmse_results %>% knitr::kable()
```

Model	RMSE
Just the average	1.061202

Model 2 (Movie Effects)

Our second model will implement movie averages from the residuals. Our RMSE is 0.9439087.

```
movie_avg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating-mu))

prediction <- validation %>%
  left_join(movie_avg, by = "movieId") %>%
```

```
  mutate(prediction = mu + b_i) %>%
  pull(prediction)

rmse_results <- bind_rows(rmse_results,
                          tibble(Model = "Movie Effect",
                                 RMSE = RMSE(validation$rating, prediction)))

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

Model	RMSE
Just the average	1.0612018
Movie Effect	0.9439087

Model 3 (Movie + User Effects)

Our third model will integrate users' averages. Some users are generous in giving ratings, but some are not. Our RMSE will be 0.8653488.

```
user_avg <- edx %>%
  left_join(movie_avg, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

prediction <- validation %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  mutate(prediction = mu + b_i + b_u) %>%
  pull(prediction)

rmse_results <- bind_rows(rmse_results,
                          tibble(Model = "Movie + User Effects",
                                 RMSE = RMSE(validation$rating, prediction)))

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

Model	RMSE
Just the average	1.0612018
Movie Effect	0.9439087
Movie + User Effects	0.8653488

Model 4 (Movie + User + Timestamp (Day) Effects)

We will round the timestamp to the nearest day and find their averages, apply these averages to our model, and get an RMSE of 0.8648351.

```
# Mutate edx and validation
edx <- edx %>%
  mutate(day = timestamp %>% as_datetime() %>% round_date(unit = "day"),
         year = timestamp %>% as_datetime() %>% round_date(unit = "year"))
validation <- validation %>%
  mutate(day = timestamp %>% as_datetime() %>% round_date(unit = "day"),
         year = timestamp %>% as_datetime() %>% round_date(unit = "year"))
```

```r
# Timestamp effect
day_avg <- edx %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  group_by(day) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u))
prediction <- validation %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  left_join(day_avg, by = "day") %>%
  mutate(prediction = mu + + b_i + b_u + b_t) %>%
  pull(prediction)
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = "Movie + User + Timestamp (Day) Effects",
                                 RMSE = RMSE(validation$rating, prediction)))

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

Model	RMSE
Just the average	1.0612018
Movie Effect	0.9439087
Movie + User Effects	0.8653488
Movie + User + Timestamp (Day) Effects	0.8648351

Model 5 (Movie + User + Timestamp + Release Date Effects)

First, we will extract the release date from the *title* variable. Next, we will find the averages for these release dates and find implement them into our model. Our RMSE is 0.8644895.

```r
# Release date effect

# Extracting release date
release_date <- edx$title %>%
  str_extract(pattern = "\\(\\d{4}\\)$") %>%
  str_replace(pattern = "\\(", replacement = "") %>%
  str_replace(pattern = "\\)", replacement = "")
edx <- edx %>%
  mutate(release_date = release_date)
release_date <- validation$title %>%
  str_extract(pattern = "\\(\\d{4}\\)$") %>%
  str_replace(pattern = "\\(", replacement = "") %>%
  str_replace(pattern = "\\)", replacement = "")
validation <- validation %>%
  mutate(release_date = release_date)
rm(release_date)

# Prediction
release_avg <- edx %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  left_join(day_avg, by = "day") %>%
  group_by(release_date) %>%
```

```
  summarize(b_rel = mean(rating-mu-b_i-b_u-b_t))
prediction <- validation %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  left_join(day_avg, by = "day") %>%
  left_join(release_avg, by = "release_date") %>%
  mutate(prediction = mu + b_i + b_u + b_t + b_rel) %>%
  pull(prediction)
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = str_c("Movie + User + Timestamp (Day) ",
                                               "+ Release Date Effects"),
                                 RMSE = RMSE(validation$rating, prediction)))

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

Model	RMSE
Just the average	1.0612018
Movie Effect	0.9439087
Movie + User Effects	0.8653488
Movie + User + Timestamp (Day) Effects	0.8648351
Movie + User + Timestamp (Day) + Release Date Effects	0.8644895

Model 6 (Movie + User + Timestamp + Release Date + Age Effects)

We will gather age upon rating data by subtracting the timestamp with the release date. This method will mean that our age will be rounded to the nearest year because our release dates are in years. Incorporating the release date averages into our model, our RMSE will be 0.8641410.

```
# Age effect

# By year
age <- as.Date(edx$year) - as.Date(edx$release_date, format = "%Y")
edx <- edx %>%
  mutate(age_year = age)
age <- as.Date(validation$year) - as.Date(validation$release_date, format = "%Y")
validation <- validation %>%
  mutate(age_year = age)
rm(age)
age_avg <- edx %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  left_join(day_avg, by = "day") %>%
  left_join(release_avg, by = "release_date") %>%
  group_by(age_year) %>%
  summarize(b_age = mean(rating - mu - b_i - b_u - b_t - b_rel))
prediction <- validation %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  left_join(day_avg, by = "day") %>%
  left_join(release_avg, by = "release_date") %>%
  left_join(age_avg, by = "age_year") %>%
  mutate(prediction = mu + b_i + b_u + b_t + b_rel + b_age) %>%
  pull(prediction)
```

```
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = str_c("Movie + User + Timestamp (Day) ",
                                               "+ Release Date + Age (Year) Effects"),
                                 RMSE = RMSE(validation$rating, prediction)))

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

Model	RMSE
Just the average	1.0612018
Movie Effect	0.9439087
Movie + User Effects	0.8653488
Movie + User + Timestamp (Day) Effects	0.8648351
Movie + User + Timestamp (Day) + Release Date Effects	0.8644895
Movie + User + Timestamp (Day) + Release Date + Age (Year) Effects	0.8641410

Model 7 (Movie + User + Timestamp + Release Date + Age + Genre Effects)

We will apply combinations of genres averages into our model. Doing so will explain much more information than splitting the genres. As a result, produce a better RMSE. Our RMSE for this model is 0.8638215.

```
genre_avg <- edx %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  left_join(day_avg, by = "day") %>%
  left_join(release_avg, by = "release_date") %>%
  left_join(age_avg, by = "age_year") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_t - b_rel - b_age))
prediction <- validation %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  left_join(day_avg, by = "day") %>%
  left_join(release_avg, by = "release_date") %>%
  left_join(age_avg, by = "age_year") %>%
  left_join(genre_avg, by = "genres") %>%
  mutate(prediction = mu + b_i + b_u + b_t + b_rel + b_age + b_g) %>%
  pull(prediction)
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = str_c("Movie + User + Timestamp (Day) ",
                                               "+ Release Date + Age (Year) + Genre Effects"),
                                 RMSE = RMSE(validation$rating, prediction)))

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

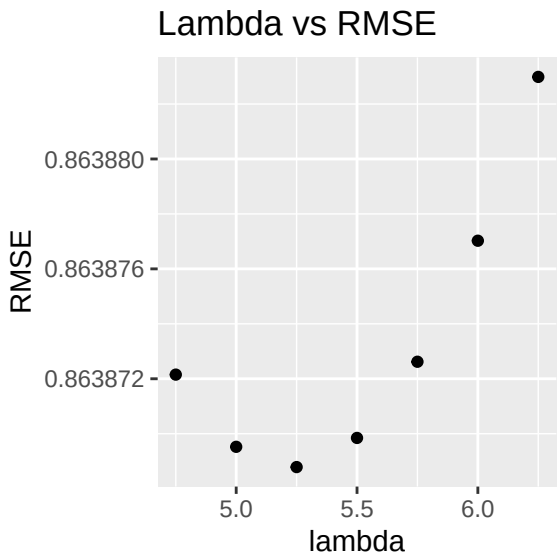
Model	RMSE
Just the average	1.0612018
Movie Effect	0.9439087
Movie + User Effects	0.8653488
Movie + User + Timestamp (Day) Effects	0.8648351
Movie + User + Timestamp (Day) + Release Date Effects	0.8644895
Movie + User + Timestamp (Day) + Release Date + Age (Year) Effects	0.8641410
Movie + User + Timestamp (Day) + Release Date + Age (Year) + Genre Effects	0.8638215

Model 8 (Regularization Movie + User + Timestamp + Release Date + Age + Genres Effects)

Because some averages are derived from a few numbers of entries, we must incorporate regularization in our model to negate this. The parameter used for regularization is *lambda*. We will use cross-validation to find *lambda* that produces the best RMSE. Firstly, we will divide *edx* into *train_set* and *test_set*. The code below will first cross-validate on lambda using *train_set*. Furthermore, it will use cross-validation on *edx* to estimate the RMSE for the final prediction. Lastly, the code will perform the final prediction on *validation* and store the results on *rmse_results*. The final results will show that the actual result at *rmse_results* does not differ that much from our estimation *cv_estimate*. Our RMSE will be 0.8632778.

```
k <- 25
lambda <- seq(4.75,6.25,0.25)

set.seed(1)
test_index <- createDataPartition(edx$rating, times = 1, p = 0.1, list = FALSE)
temp <- edx %>% slice(test_index)
train_set <- edx %>% slice(-test_index)
# Make sure movieId, userId, day, release date, age, 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 = "day") %>%
  semi_join(train_set, by = "release_date") %>%
  semi_join(train_set, by = "age_year") %>%
  semi_join(train_set, by = "genres")
# Add rows removed from train set back into edx set
removed <- anti_join(temp, test_set) # Removed 17 rows only
train_set <- bind_rows(train_set, removed)
rm(test_index, removed, temp)

#Create train_set partitions
set.seed(1)
folds_index <- createFolds(train_set$rating, k = k)

# Cross validation (Finding the best lambda for regularization)
train_folds_index <- seq(1:nrow(train_set))
best_lambda <- function(lambda){
  sapply(1:k, function(x){
    index <- train_folds_index
    index <- index[!index %in% folds_index[[x]]]
    train_folds_set <- train_set[index,]
    temp <- train_set[(folds_index[[x]]),]
    # Make sure movieId, userId, timestamp (day), release date, age, and genres are in test set are als
    test_folds_set <- temp %>%
      semi_join(train_folds_set, by = "movieId") %>%
      semi_join(train_folds_set, by = "userId") %>%
      semi_join(train_folds_set, by = "day") %>%
      semi_join(train_folds_set, by = "release_date") %>%
      semi_join(train_folds_set, by = "age_year") %>%
      semi_join(train_folds_set, by = "genres")
    # Add rows removed from test set back into train set
    removed <- anti_join(temp, test_folds_set)
    train_folds_set <- bind_rows(train_folds_set, removed)
    rm(index, temp, removed)
```

```
    movie_avg <- train_folds_set %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating-mu)/(n() + lambda))
    temp <- train_folds_set %>%
      left_join(movie_avg, by = "movieId")
    user_avg <- temp %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating-mu-b_i)/(n() + lambda))
    temp <- temp %>%
      left_join(user_avg, by = "userId")
    time_avg <- temp %>%
      group_by(day) %>%
      summarize(b_t = sum(rating - mu - b_i - b_u)/(n() + lambda))
    temp <- temp %>%
      left_join(time_avg, by = "day")
    release_avg <- temp %>%
      group_by(release_date) %>%
      summarize(b_rel = sum(rating - mu - b_i - b_u - b_t)/(n() + lambda))
    temp <- temp %>%
      left_join(release_avg, by = "release_date")
    age_avg <- temp %>%
      group_by(age_year) %>%
      summarize(b_age = sum(rating - mu - b_i - b_u - b_t - b_rel)/(n() + lambda))
    temp <- temp %>%
      left_join(age_avg, by = "age_year")
    genre_avg <- temp %>%
      group_by(genres) %>%
      summarize(b_g = sum(rating- mu - b_i - b_u - b_t - b_rel - b_age)/(n() + lambda))
    rm(temp)

    prediction <- test_folds_set %>%
      left_join(movie_avg, by = "movieId") %>%
      left_join(user_avg, by = "userId") %>%
      left_join(time_avg, by = "day") %>%
      left_join(release_avg, by = "release_date") %>%
      left_join(age_avg, by = "age_year") %>%
      left_join(genre_avg, by = "genres") %>%
      mutate(prediction = mu + b_i + b_u + b_t + b_rel + b_age + b_g) %>%
      pull(prediction)

    return(RMSE(test_folds_set$rating, prediction))

  })
}

result <- sapply(lambda, best_lambda)
# Removing unsued variables
rm(folds_index, train_folds_index)
# PLot RMSE against lambda
ggplot(mapping = aes(lambda, colMeans(result))) +
  geom_point() +
  ggtitle("Lambda vs RMSE") +
  ylab("RMSE")
```




```
#Best Lambda
index <- colMeans(result) %>% which.min()
lambda <- lambda[index]
lambda
```

```
## [1] 5.25
```

```
rm(index)


# Estimating RMSE with the test set

k <- 10
lambda <- 5.25

folds_index <- createFolds(edx$rating, k = k)
result <- sapply(1:k, function(x){
  temp <- edx %>% slice(folds_index[[x]])
  train_set <- edx %>% slice(-folds_index[[x]])
  # Make sure movieId, userId, day, release date, age, and genres in the test set are
  # also in the train set
  test_set <- temp %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId") %>%
    semi_join(train_set, by = "day") %>%
    semi_join(train_set, by = "release_date") %>%
    semi_join(train_set, by = "age_year") %>%
    semi_join(train_set, by = "genres")
  # Add rows removed from test set back into train set
  removed <- anti_join(temp, test_set)
  train_set <- bind_rows(train_set, removed)
  rm(temp, removed)

  movie_avg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating-mu)/(n() + lambda))
  temp <- train_set %>%
```

```
    left_join(movie_avg, by = "movieId")
  user_avg <- temp %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating-mu-b_i)/(n() + lambda))
  temp <- temp %>%
    left_join(user_avg, by = "userId")
  time_avg <- temp %>%
    group_by(day) %>%
    summarize(b_t = sum(rating - mu - b_i - b_u)/(n() + lambda))
  temp <- temp %>%
    left_join(time_avg, by = "day")
  release_avg <- temp %>%
    group_by(release_date) %>%
    summarize(b_rel = sum(rating - mu - b_i - b_u - b_t)/(n() + lambda))
  temp <- temp %>%
    left_join(release_avg, by = "release_date")
  age_avg <- temp %>%
    group_by(age_year) %>%
    summarize(b_age = sum(rating - mu - b_i - b_u - b_t - b_rel)/(n() + lambda))
  temp <- temp %>%
    left_join(age_avg, by = "age_year")
  genre_avg <- temp %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating- mu - b_i - b_u - b_t - b_rel - b_age)/(n() + lambda))
  rm(temp)

  prediction <- test_set %>%
    left_join(movie_avg, by = "movieId") %>%
    left_join(user_avg, by = "userId") %>%
    left_join(time_avg, by = "day") %>%
    left_join(release_avg, by = "release_date") %>%
    left_join(age_avg, by = "age_year") %>%
    left_join(genre_avg, by = "genres") %>%
    mutate(prediction = mu + b_i + b_u + b_t + b_rel + b_age + b_g) %>%
    pull(prediction)

  return(RMSE(test_set$rating, prediction))
})
rm(folds_index)
cv_estimate <- tibble()
cv_estimate <- bind_rows(cv_estimate,
                         tibble(Model = "(Model 8) CV Estimate",
                                RMSE = mean(result)))

# Final prediction
lambda <- 5.25
movie_avg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating-mu)/(n() + lambda))
temp <- edx %>%
  left_join(movie_avg, by = "movieId")
user_avg <- temp %>%
  group_by(userId) %>%
```

```
  summarize(b_u = sum(rating-mu-b_i)/(n() + lambda))
temp <- temp %>%
  left_join(user_avg, by = "userId")
time_avg <- temp %>%
  group_by(day) %>%
  summarize(b_t = sum(rating - mu - b_i - b_u)/(n() + lambda))
temp <- temp %>%
  left_join(time_avg, by = "day")
release_avg <- temp %>%
  group_by(release_date) %>%
  summarize(b_rel = sum(rating - mu - b_i - b_u - b_t)/(n() + lambda))
temp <- temp %>%
  left_join(release_avg, by = "release_date")
age_avg <- temp %>%
  group_by(age_year) %>%
  summarize(b_age = sum(rating - mu - b_i - b_u - b_t - b_rel)/(n() + lambda))
temp <- temp %>%
  left_join(age_avg, by = "age_year")
genre_avg <- temp %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating- mu - b_i - b_u - b_t - b_rel - b_age)/(n() + lambda))
rm(temp)

prediction <- validation %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  left_join(time_avg, by = "day") %>%
  left_join(release_avg, by = "release_date") %>%
  left_join(age_avg, by = "age_year") %>%
  left_join(genre_avg, by = "genres") %>%
  mutate(prediction = mu + b_i + b_u + b_t + b_rel + b_age + b_g) %>%
  pull(prediction)

rmse_results <- bind_rows(rmse_results,
                          tibble(Model = str_c("Regularization Movie + User ",
                                               "+ Timestamp (Day) + Release Date + Age (Year) ",
                                               " + Genres Effects"),
                                 RMSE = RMSE(validation$rating, prediction)))

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

Model	RMSE
Just the average	1.0612018
Movie Effect	0.9439087
Movie + User Effects	0.8653488
Movie + User + Timestamp (Day) Effects	0.8648351
Movie + User + Timestamp (Day) + Release Date Effects	0.8644895
Movie + User + Timestamp (Day) + Release Date + Age (Year) Effects	0.8641410
Movie + User + Timestamp (Day) + Release Date + Age (Year) + Genre Effects	0.8638215
Regularization Movie + User + Timestamp (Day) + Release Date + Age (Year) + Genres Effects	0.8632778

Model 9 (Matrix Factorization with Best Basline Model)

Matrix factorization will further improve our RMSE significantly. It will explain information regarding the correlation between groups of movies and groups of users. Matrix factorization will explain information that basic predictors just cannot do. To do this, we will use the *recosystem* package. Our RMSE for the final and best model is 0.7860442.

```
library(recosystem)
#Residual data
lambda <- 5.25
movie_avg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating-mu)/(n() + lambda))
temp <- edx %>%
  left_join(movie_avg, by = "movieId")
user_avg <- temp %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating-mu-b_i)/(n() + lambda))
temp <- temp %>%
  left_join(user_avg, by = "userId")
time_avg <- temp %>%
  group_by(day) %>%
  summarize(b_t = sum(rating - mu - b_i - b_u)/(n() + lambda))
temp <- temp %>%
  left_join(time_avg, by = "day")
release_avg <- temp %>%
  group_by(release_date) %>%
  summarize(b_rel = sum(rating - mu - b_i - b_u - b_t)/(n() + lambda))
temp <- temp %>%
  left_join(release_avg, by = "release_date")
age_avg <- temp %>%
  group_by(age_year) %>%
  summarize(b_age = sum(rating - mu - b_i - b_u - b_t - b_rel)/(n() + lambda))
temp <- temp %>%
  left_join(age_avg, by = "age_year")
genre_avg <- temp %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating- mu - b_i - b_u - b_t - b_rel - b_age)/(n() + lambda))
rm(temp)

residual <- edx %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  left_join(time_avg, by = "day") %>%
  left_join(release_avg, by = "release_date") %>%
  left_join(age_avg, by = "age_year") %>%
  left_join(genre_avg, by = "genres") %>%
  mutate(residual = rating - mu - b_i - b_u - b_t - b_rel - b_age - b_g) %>%
  select(userId, movieId, residual)

#Matrix Factorization (Recosystem package)
edx_mf <- as.matrix(residual)
validation_mf <- validation %>% select(userId, movieId, rating) %>% as.matrix()

#Storing data in hard drive
```

```
write.table(edx_mf, file = "train.txt", sep = " ", row.names = FALSE, col.names = FALSE)
write.table(validation_mf, file = "test.txt", sep = " ", row.names = FALSE, col.names = FALSE)
set.seed(1)
train_data <- data_file("train.txt")
validation_data <- data_file("test.txt")
rm(edx_mf, validation_mf)

#Creating model object
r <- Reco()
#Tuning parameters
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                       costp_l1 = 0, costq_l1 = 0,
                                       nthread = 1, niter = 10))
r$train(train_data, opts = c(opts$min, nthread = 1, niter = 20))
stored_prediction <- tempfile()
r$predict(validation_data, out_file(stored_prediction))
pred_ratings <- scan(stored_prediction)
#Making final prediction
temp <- validation %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  left_join(time_avg, by = "day") %>%
  left_join(release_avg, by = "release_date") %>%
  left_join(age_avg, by = "age_year") %>%
  left_join(genre_avg, by = "genres") %>%
  mutate(prediction = mu + b_i + b_u + b_t + b_rel + b_age + b_g) %>%
  pull(prediction)
prediction <- pred_ratings + temp
rm(temp)
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = "Matrix Factorization with Best Baseline Model",
                                 RMSE = RMSE(validation$rating, prediction)))
#Removing unused variables
rm(train_data, validation_data, r, opts, stored_prediction)
```

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

Model	RMSE
Just the average	1.0612018
Movie Effect	0.9439087
Movie + User Effects	0.8653488
Movie + User + Timestamp (Day) Effects	0.8648351
Movie + User + Timestamp (Day) + Release Date Effects	0.8644895
Movie + User + Timestamp (Day) + Release Date + Age (Year) Effects	0.8641410
Movie + User + Timestamp (Day) + Release Date + Age (Year) + Genre Effects	0.8638215
Regularization Movie + User + Timestamp (Day) + Release Date + Age (Year) + Genres Effects	0.8632778
Matrix Factorization with Best Baseline Model	0.7860442

Conclusion

From the 9 models deduced above, we have the RMSE results as follows:

Model	RMSE
Just the average	1.0612018
Movie Effect	0.9439087
Movie + User Effects	0.8653488
Movie + User + Timestamp (Day) Effects	0.8648351
Movie + User + Timestamp (Day) + Release Date Effects	0.8644895
Movie + User + Timestamp (Day) + Release Date + Age (Year) Effects	0.8641410
Movie + User + Timestamp (Day) + Release Date + Age (Year) + Genre Effects	0.8638215
Regularization Movie + User + Timestamp (Day) + Release Date + Age (Year) + Genres Effects	0.8632778
Matrix Factorization with Best Baseline Model	0.7860442

The results show that the best model is Model 9 (Matrix Factorization with Best Basline Model) with a RMSE of 0.7860442.

MovieLens is a traditional dataset for proposal framework and speaks to a test for improvement of better AI calculation. In our first model, "Just the average", our RMSE is 1.0612018. The first model is the most basic prediction algorithm. Our final model has an RMSE of 0.7860442. All in all, matrix factorization has all the earmarks of being an exceptionally ground-breaking method for recommendation systems, which for the most part contains enormous and sparse datasets, making it difficult to make predictions utilizing other methods. The results above show the two most basic predictors, which are movie and user effects drops the RMSE significantly, and they explain most of the information quite well. The other predictors are integrated to slightly improve the RMSE.

Few things can be done to improve the final and best model above. These are all still hypotheses. First, the idea that user personality is different with different combination of genres and therefore, correlate to the generosity in giving ratings. Second, the idea that in certain years, certain genres are more popular and more high quality, therefore, producing a high rating. In other words, in some eras, movies are more focused to a specific style and genre where most people are most attracted to. Lastly, the fact that from data exploration, the more often a movie is rated, the higher the rating is. Nevertheless, there are numerous ways to deal with this recommendation system in which our creativity is the only limit into finding the best solution.