MovieLens- RMSE Project

Orel Adani

2023-09-07

1 Introduction

The Movielens project aims to predict movie ratings based on available features and demonstrate knowledge and skills learned during the HarvardX Professional Certificate in Data Science program. The MovieLens dataset contains user ratings for movies from 1 to 5 stars (includes half starts rating). It is provided by GroupLens research center at Minnesota University. Using these predicted ratings, companies can provide high predicted rating for their users. A large number of user ratings is needed to develop a recommendation system with meaningful accuracy.

The goal is to customizing this recommendations to the users by provide them to watch more movies to their liking. This report sets out the exploratory analysis of the data using common visualization techniques followed by the methods used to develop, train and test the algorithm before providing and discussing the results from each iteration of the algorithm and concluding on the outcome of the final model, its limitations and potential for future work.

Creating the “Test” and “Training” sets

Partitioning the data into 2 sets: 90% of the data will be in the edx data set, and 10% in the Final Holdout Test set. Adding mutating fields to edx1 for predictive purposes and for explanatory reasons.

```
# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later

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

# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in
edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

```
# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)

## Joining with `by = join_by(userId, movieId, rating, timestamp,
title, genres)`

edx <- rbind(edx, removed)

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

edx1 <- edx %>% mutate(ratingdate=as_date(as_datetime(timestamp)),
                       ratingyear=as.numeric(format(ratingdate,"%Y")),
                       ratingmonth=as.numeric(format(ratingdate,"%m")),
                       releaseyear =
as.numeric(str_extract(str_extract(title, "[/(]\\d{4}[/)]$"),
regex("\\d{4}")),title = str_remove(title, "[/(]\\d{4}[/)]$")),
                       nostalgialevel=ratingyear-releaseyear,
                       nostalgiasplit=case_when(nostalgialevel >=10 ~
"NostalgialMovies",TRUE ~"inDecade"),
                       release_year_era=case_when(releaseyear >=1910 &
releaseyear <=1919 ~  "1910s",              # define the decades of
the release date
                                                  releaseyear >=1920 &
releaseyear <=1929~  "1920s",
                                                  releaseyear >=1930 &
releaseyear <=1939 ~  "1930s",
                                                  releaseyear >=1940 &
releaseyear <=1949 ~  "1940s",
                                                  releaseyear >=1950 &
releaseyear <=1959 ~  "1950s",
                                                  releaseyear >=1960 &
releaseyear <=1969 ~  "1960s",
                                                  releaseyear >=1970 &
releaseyear <=1979~  "1970s",
                                                  releaseyear >=1980 &
releaseyear <=1989 ~  "1980s",
                                                  releaseyear >=1990 &
releaseyear <=1999 ~  "1990s",
                                                  releaseyear >=2000 &
releaseyear <=2009~  "2000s",
                                                  TRUE ~"2010s"),
                       release_year_era_group=case_when (releaseyear
>=1910 & releaseyear <=1979~ "1970s & earlier",       # group the
decades to : prior 1980s, 1980s,1990s,2000s
                                                        releaseyear
>=1980 & releaseyear <=1989~ "1980s",
                                                        releaseyear
>=1990 & releaseyear <=1999~ "1990s",
                                                        releaseyear
>=2000 & releaseyear <=2009~ "2000s" ,
                                                    TRUE ~"2010s")
                      )
```

Splitting edx1 out into train and test sets

As the final_holdout_test data set was reserved for the final hold-out test, the edx1 data set needed to be used both to train and test the algorithm in development. This is important to allow for cross-validation and refinement of the final model without the risk of over-training. Here, the same technique was applied as with the original movielens data set, dividing the edx data set into train (90%) and test (10%) sets. As before, lets ensure that the test set only included users and movies that are present in the train set and, secondly adding the removed data to the train set in order to maximise the data available for training purposes.

```
# Create train set and test sets from edx1
# set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
 test_index <- createDataPartition(y = edx1$rating, times = 1, p = 0.1,
list = FALSE)
 train_set <- edx1[-test_index,]
 temp <- edx1[test_index,]

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

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

## Joining with `by = join_by(userId, movieId, rating, timestamp,
title, genres,
## ratingdate, ratingyear, ratingmonth, releaseyear, nostalgialevel,
## nostalgiasplit, release_year_era, release_year_era_group)`

 train_set <- rbind(train_set, removed)

 rm( test_index, temp, removed)
```

2 Analysis

Data Exploration

Preview of the data set:

```
head(movielens)

##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046                Boomerang (1992)
## 2      1     185      5 838983525                 Net, The (1995)
## 3      1     231      5 838983392            Dumb & Dumber (1994)
## 4      1     292      5 838983421                 Outbreak (1995)
## 5      1     316      5 838983392                 Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 3                        Comedy
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
```

That produce 10M rows(as the number of ratings) and 6 columns:

```
dim(movielens)

## [1] 10000054        6
```

And create:

```
movielens %>%
  summarise(n_movies = n_distinct(movieId),
            n_users = n_distinct(userId))

##   n_movies n_users
## 1    10677   69878
```

The prediction of rating will be evaluated using the residual mean squared error (RMSE):

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

$$N$$

is defined as the number of user/movie combinations,

$$y_{u,i}$$

as the rating for movie i by user u .

The RMSE is a commonly measures the differences between predicted and observed values. It can be interpreted similarly to a standard deviation. For this exercise if the number is larger than 1 it means our typical error is larger than one star. The goal is to reduce this error by adding more explanatory predictors.

The prediction:

- **rating** is a value from 0.5 to 5 provided by user *U* for movie *i* ;

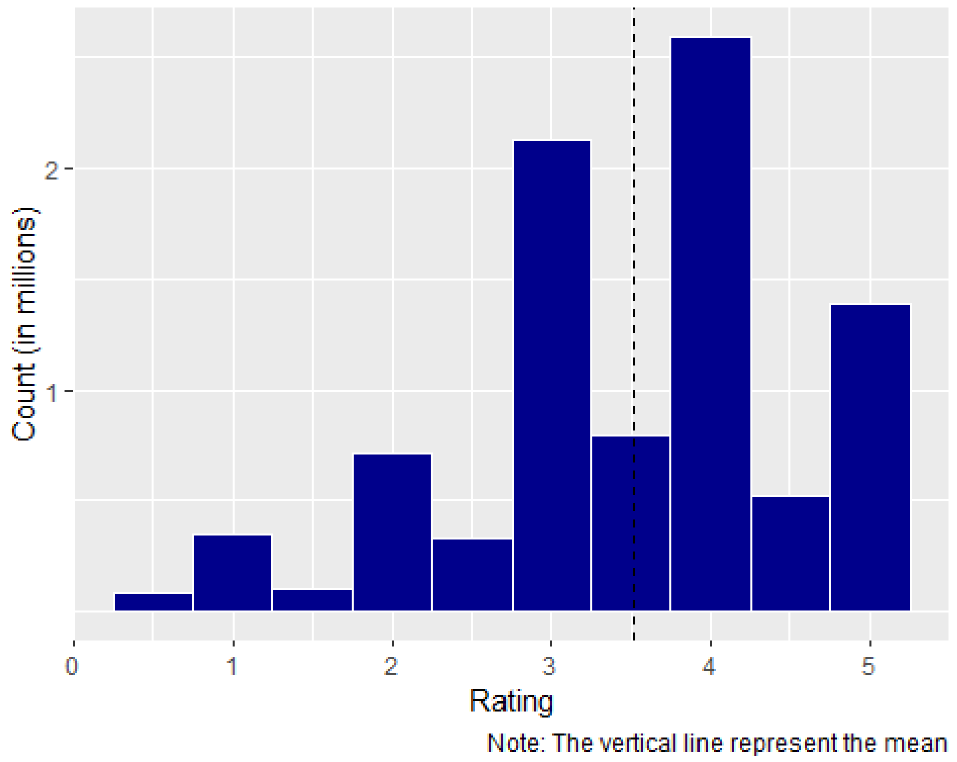
The predictors are:

- **movieId** is a unique movie identification.(Will use title for visualization porpoises)
- **userId** is a unique user identification.
- **timestamp** is a date and time of the rating- Will use the year date only.
- **Release Year** is Excluded from the movies title.
- **genres** is a movie genre.
- **nostalgia level** is the spared between rating year and release year.

2.1 The prediction: Rating

The overall average rating in the edx1 data set was 3.51. The minimum rating awarded to any movie was 0.5 and the maximum rating awarded was 5. The distribution of total ratings included in the data set shows that the most common rating across all movies was 4, and that overall, whole star ratings (7,156,885;79.5%) were used more than half star ratings (1,843,170; 20.5%).

```
 edx1 %>% ggplot(aes(rating)) +
   geom_histogram(bins=10,col="white", fill = "darkblue") +
   geom_vline(xintercept = mean(edx1$rating), lty = 2) +
   scale_y_continuous(breaks = c(1000000, 2000000), labels = c("1",
"2")) +
   labs(x = "Rating", y = "Count (in millions)", caption = "Note: The
vertical line represent the mean")
```



2.1.1 Prediction Based on Mean Rating

This model uses the simple mean of the data set to predict the rating for all movies, the model assumes that all differences are due to a random error.

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

where μ the expected rating and $\epsilon_{u,i}$ is the independent errors across all movies.

```
  # Calculate the overall average rating across all movies included in
train set
```

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

 mu_hat

## [1] 3.512509

 # Calculate RMSE between each rating included in test set and the
overall average
 simple_rmse <- RMSE(test_set$rating, mu_hat)


# Save Results to a Table
rmse_results = tibble(Method = "Naive Analysis by Mean", RMSE =
simple_rmse)
rmse_results %>% knitr::kable()
```

Method	RMSE
Naive Analysis by Mean	1.061135

This prediction gives us a rather high RMSE over 1 which is far from ideal and doesn't help us in building a recommendation system. Now, we try to use variables to get a better working system.

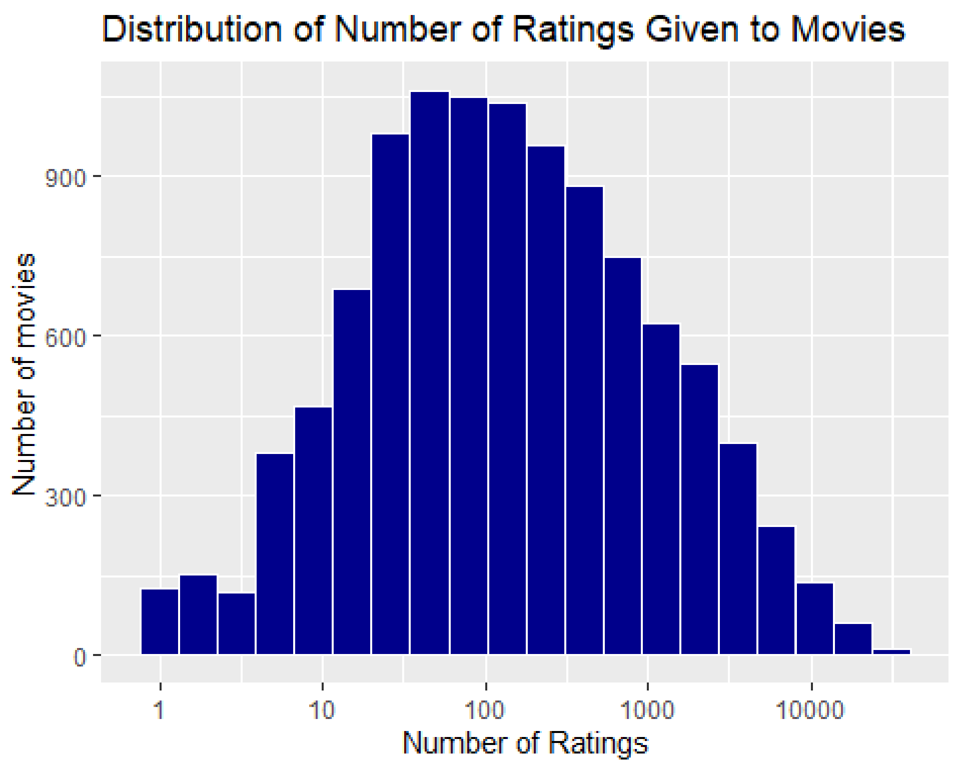
2.2 Predictor 1-2: Movies & Users

Examine the variance in the number of ratings given by each movie, and the number of ratings given to each user. This will help our approach during model development.

Movies

We will look at the number of ratings received by each movie:

```
# histogram of distribution of movie ratings
edx1 %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins=20,col="white", fill = "darkblue") +
  scale_x_log10() +
  ggtitle("Distribution of Number of Ratings Given to Movies") +
  xlab("Number of Ratings") +
  ylab("Number of movies")
```



The Movie Effects Model predicting the movie ratings with both biases gives an improved prediction with a lower RMSE value.

```
movies_count_top_m<-edx1 %>%

group_by(title) %>%
summarize(n_rating = n(),

mu_movie = mean(rating),

sd_movie = sd(rating)) %>%

arrange(desc(n_rating))

movies_count_bottom_m<-edx1 %>%

group_by(title) %>%

summarize(n_rating = n(),

mu_movie = mean(rating),

sd_movie = sd(rating)) %>%

arrange((n_rating))

movies_count_top_m[1:5,]

## # A tibble: 5 × 4
##   title                                n_rating mu_movie sd_movie
##   <chr>                                   <int>    <dbl>    <dbl>
## 1 Pulp Fiction (1994)                     31362     4.15    1.00
## 2 Forrest Gump (1994)                     31079     4.01    0.972
## 3 Silence of the Lambs, The (1991)        30382     4.20    0.839
## 4 Jurassic Park (1993)                    29360     3.66    0.938
## 5 Shawshank Redemption, The (1994)        28015     4.46    0.717

movies_count_bottom_m[1:5,]

## # A tibble: 5 × 4
##   title                                             n_rating mu_movie
sd_movie
##   <chr>                                                <int>    <dbl>
<dbl>
## 1 1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)            1      2
NA
## 2 100 Feet (2008)                                          1      2
NA
## 3 4 (2005)                                                 1      2.5
NA
## 4 Accused (Anklaget) (2005)                                1      0.5
NA
## 5 Ace of Hearts (2008)                                     1      2
NA
```

Looking at the top 5 and bottom 5 movies rating we understand We cannot compare ratings of a movies equally. Further analysis reveals significant variation in the number of ratings received by each movie, with the movie with the most ratings, Pulp Fiction (1994), receiving a total of 31362 ratings where 126 movies were only rated once. There is clearly a movie effect on the rating worthwhile to include in the training algorithm.

2.2.1 Movie Effects Model

We saw that there's a movie bias with some movies being rated higher that others and we try to incorporate this basis into our algorithm. We calculate the movie bias as the difference between the movie's mean rating vs the overall mean rating obtained earlier.

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

where b_i is the movie bias for each movie i.

```r
# Estimate movie effect (b_i)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu_hat))
# Predict ratings adjusting for movie effects
predicted_b_i <- mu_hat + test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  pull(b_i)
# Calculate RMSE based on movie effects model
movie_rmse <- RMSE(predicted_b_i, test_set$rating)
```

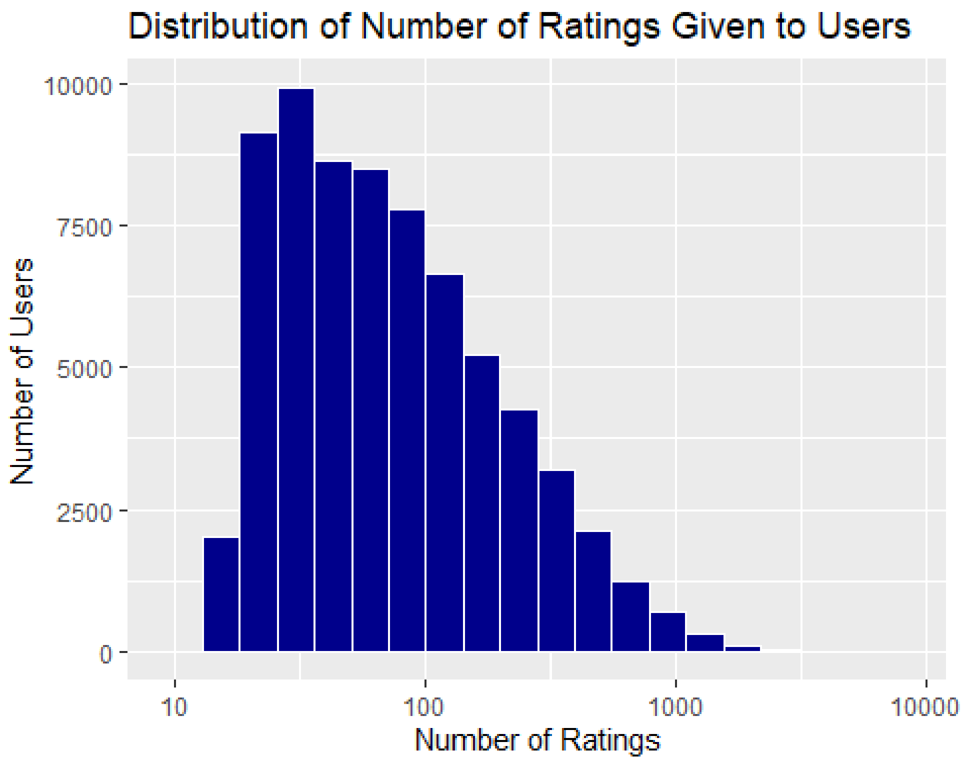
```r
# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Movie Effects Model",
                          RMSE = movie_rmse,
                          Prev_Diff=round(movie_rmse-simple_rmse, 5)))
rmse_results %>% knitr::kable()
```

Method	RMSE	Prev_Diff
Naive Analysis by Mean	1.0611350	NA
Movie Effects Model	0.9441568	-0.11698

User

Exploration of user data revealed a similar pattern to that observed for movies, with some users appearing more generous in the way they assessed movies, having provided higher ratings than others. Some users contributed many more ratings than other users This analysis identifies a clear user effect (or bias) which further improve the accuracy of a movie recommendation system.

```r
# histogram of distribution of users ratings
edx1 %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins=20,col="white", fill = "darkblue") +
  scale_x_log10() +
  ggtitle("Distribution of Number of Ratings Given to Users") +
  xlab("Number of Ratings") +
  ylab("Number of Users")
```



2.2.2 Movie and User Effects Model

We now try to use out insights on the individual User's biases to help fine tune our model acknowledging that some users will rate some movies highly while others will rate the same movie lower and vice versa. We represent this user bias as b_u into our existing model.

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

where b_u is the bias for each user u.

```
 # Estimate user effect (b_u)
user_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu_hat - b_i))
# Predict ratings adjusting for movie and user effects
predicted_b_u <- test_set %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)
# Calculate RMSE based on user effects model
user_rmse <- RMSE(predicted_b_u, test_set$rating)


# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Movie & user Effects Model",
                          RMSE = user_rmse,
                          Prev_Diff=round(user_rmse-movie_rmse, 5)))
rmse_results %>% knitr::kable()
```

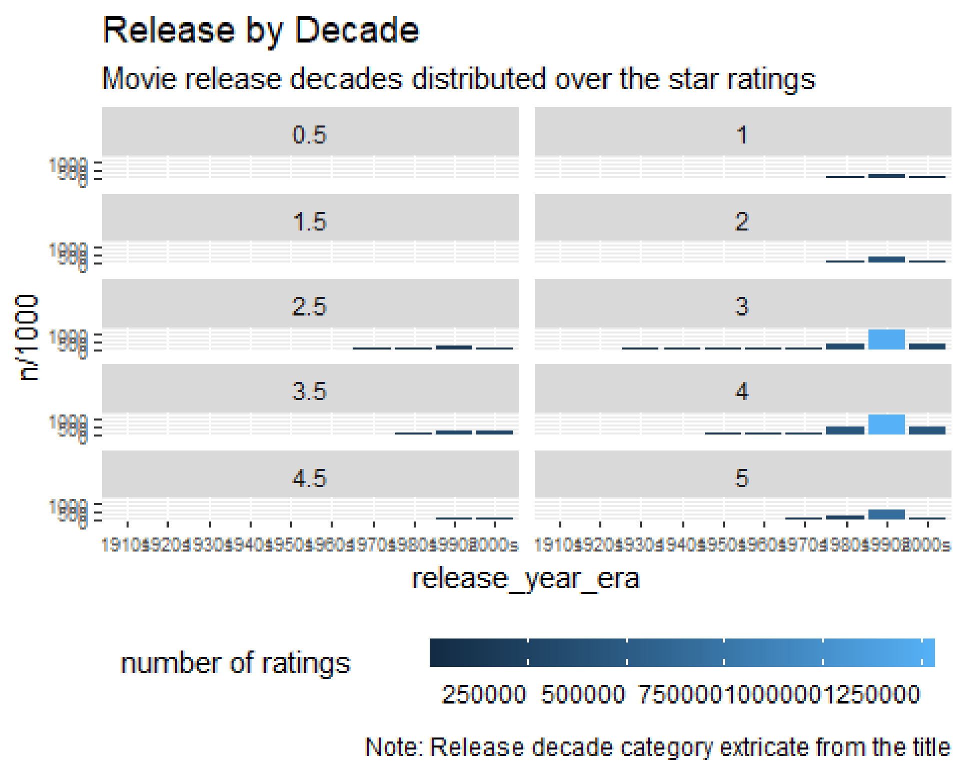
Method	RMSE	Prev_Diff
Naive Analysis by Mean	1.0611350	NA
Movie Effects Model	0.9441568	-0.11698
Movie & user Effects Model	0.8659736	-0.07818

Using other variables explored can we still try and achieve a lower RMSE?

2.3 Predictor 3: Release Year

The title variable includes both the name of the movie and the year of release, in brackets. The movies with the most ratings were released during the 1990s, peaking in 1995 with approximately 8.7% of the total number of ratings included in the edx1 data set. However, there are very few ratings within the data set assigned to movies released prior to 1970. Adjusting for the effect of release year should improve the accuracy of the training algorithm by taking to consideration different point estimates of sample sizes in some years would also be important.

```
 #decades distributed over the star ratings

   edx1 %>% group_by (rating,release_year_era) %>%
   count(release_year_era) %>%
   ggplot(aes(x = release_year_era, y = n/1000, fill= n)) +  # Plot
with values on top
   geom_bar(stat = "identity") +
   ggtitle("Release by Decade") +
   labs(fill = "number of ratings", subtitle="Movie release decades
distributed over the star ratings", caption ="Note: Release decade
category extricate from the title") +
  theme(axis.text.y = element_text(size = rel(0.75)), axis.text.x
=element_text(size = rel(0.75)), legend.position = "bottom",
legend.spacing.x = unit(1, 'cm'))+
  guides(fill = guide_colorbar(title = "number of ratings",
# edit the legend bar
                               label.position = "bottom", label.hjust =
1,
                               title.position = "left", title.vjust =
0.75,
                               barwidth = 13,
                               barheight = 0.75)) +
   facet_wrap(~ rating,ncol=2)
```



2.3.1 Movie, User and Release Year Effects Model

We now try to use out insights on the individual release year's biases to help fine tune our model acknowledging that some years releases movies more than others and the amount of users rating them changes as well. We represent this release year bias as b_y into our existing model.

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

where b_y is the bias for each release year y.

```
# Estimate release year effect (b_y)
year_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(releaseyear) %>%
  summarise(b_y = mean(rating - mu_hat - b_i - b_u  ))
# Predict ratings adjusting for movie, user and release year effects
predicted_b_y <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(year_avgs, by = "releaseyear") %>%
  mutate(pred = mu_hat + b_i + b_u + b_y) %>%
  pull(pred)
# Calculate RMSE based on year effects model
year_rmse <- RMSE(predicted_b_y, test_set$rating)
```

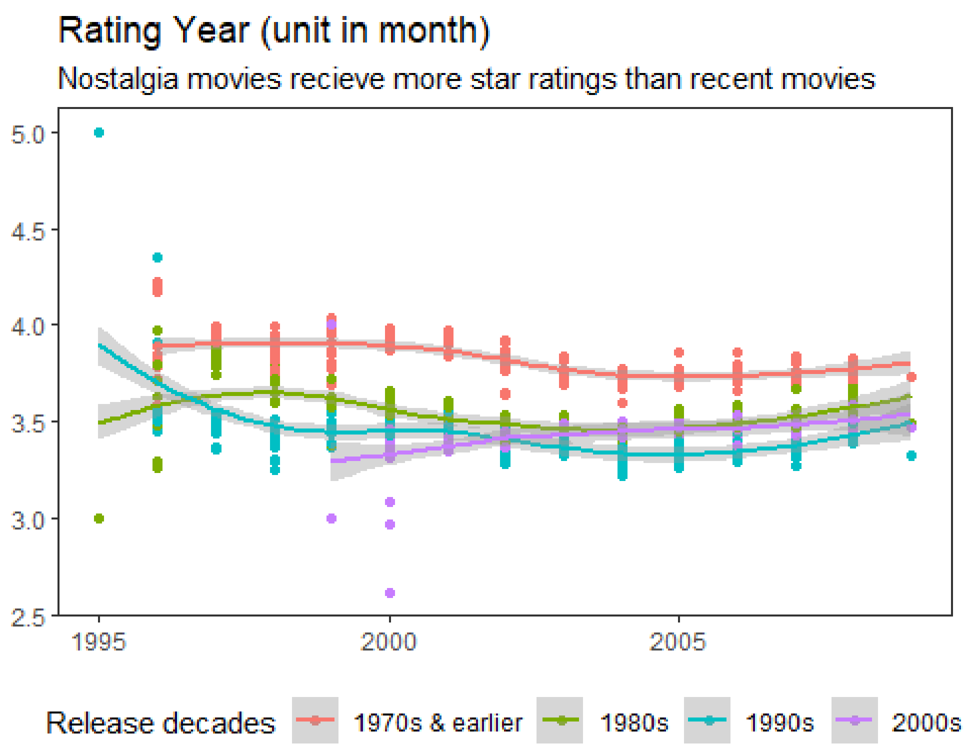
```r
# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Movie, user & release year
Effects Model",
                          RMSE = year_rmse,Prev_Diff=round(year_rmse-
user_rmse,5)))
rmse_results %>% knitr::kable()
```

Method	RMSE	Prev_Diff
Naive Analysis by Mean	1.0611350	NA
Movie Effects Model	0.9441568	-0.11698
Movie & user Effects Model	0.8659736	-0.07818
Movie, user & release year Effects Model	0.8656023	-0.00037

2.4 Predictor 4: Rating Date

To analysis the review date effect on ratings, the timestamp data was mutated into a date format allow us to get the month and the year from that filed. The earliest review included in the data set was in 1995 and the latest review was in 2009. It can be noticed that some years didn't get review rating every month. The resulting behavior teaches us to examine another variable that will shed light on their preferences.

```
 # smoothing the rating over time

 rating_decade<-edx1 %>%
   group_by(ratingyear,ratingmonth, release_year_era_group) %>%
   summarize(avgrating = mean(rating))

## `summarise()` has grouped output by 'ratingyear', 'ratingmonth'. You
can
## override using the `.groups` argument.

 #head(rating_decade)

 rating_decade%>%
   ggplot(aes(ratingyear, avgrating, colour = release_year_era_group))
+
   geom_point() +
   geom_smooth() +
   theme_bw() + theme(panel.grid = element_blank(),axis.title =
element_blank(),legend.position = "bottom") +
   labs(colour = "Release decades",
        title="Rating Year (unit in month)",
        subtitle="Nostalgia movies recieve more star ratings than
recent movies")

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



2.4.1 Movie, User, Release Year and Rating Year Effects Model

We now try to use out insights on the individual Rating year's biases to help fine tune our model acknowledging that some years the university succeeded to asks more users than others .The amount of movies rated changes as well as new movies releases to the market. We represent this release year bias as b_r into our existing model.

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

where b_r is the bias for each rating year r.

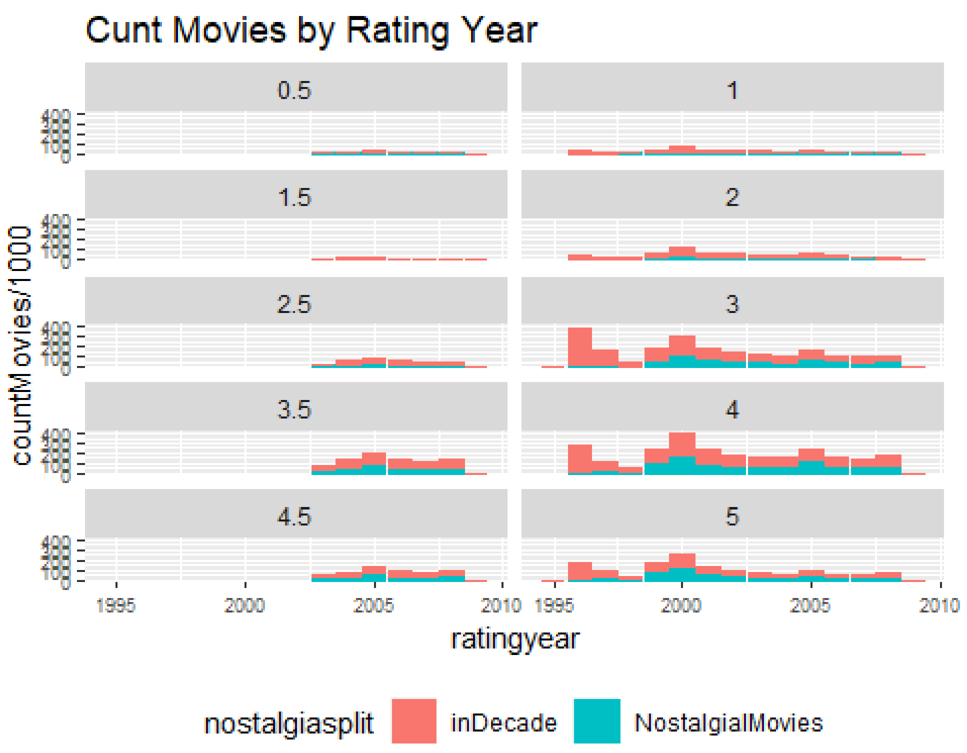
```
# Estimate rating date effect (b_r)
 date_avgs <- train_set %>%
   left_join(movie_avgs, by = "movieId") %>%
   left_join(user_avgs, by = "userId") %>%
   left_join(year_avgs, by = "releaseyear") %>%
   group_by(ratingyear) %>%
   summarise(b_r = mean(rating - mu_hat - b_i - b_u - b_y))
 # Predict ratings adjusting for movie, user, release year and rating
date effects
 predicted_b_r <- test_set %>%
   left_join(movie_avgs, by = "movieId") %>%
   left_join(user_avgs, by = "userId") %>%
   left_join(year_avgs, by = "releaseyear") %>%
   left_join(date_avgs, by = "ratingyear") %>%
   mutate(pred = mu_hat + b_i + b_u + b_y + b_r) %>%
   pull(pred)
 # Calculate RMSE based on review date effects model
 ratingdate_rmse <- RMSE(predicted_b_r, test_set$rating)

# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Movie, user, release year &
rating date Effects Model",
                          RMSE = ratingdate_rmse,
                          Prev_Diff=round(ratingdate_rmse-year_rmse,
5)))
rmse_results %>% knitr::kable()
```

Method	RMSE	Prev_Diff
Naive Analysis by Mean	1.0611350	NA
Movie Effects Model	0.9441568	-0.11698
Movie & user Effects Model	0.8659736	-0.07818
Movie, user & release year Effects Model	0.8656023	-0.00037
Movie, user, release year & rating date Effects Model	0.8655016	-0.00010

2.5 Predictor 5: Nostalgia Level

We will subtract the year of release from the year of the rating and we will get a new index that takes into account the nostalgic value the user has at the time of the rating. A movie from 1995 will be a nostalgic value to a user who rated it in 2005 compared to a user who rated it in the year of its release. Segmenting the data into Nostalgia Level groups allows us to learn about the way users preferred to rate old movies verses new ones.

```
# ratingyear distributed over the star ratings splits to nostalgia

   movies_per_year <- edx1 %>%
     select(movieId, ratingyear,nostalgiasplit,rating) %>% # select
columns we need
     group_by(ratingyear,nostalgiasplit,rating) %>% # group by year
     summarise(countMovies = n())

## `summarise()` has grouped output by 'ratingyear', 'nostalgiasplit'.
You can
## override using the `.groups` argument.

 movies_per_year %>%
   ggplot(aes(x = ratingyear, y = countMovies/1000, fill =
nostalgiasplit, colour = nostalgiasplit)) +
  geom_bar(stat = 'identity') +
   ggtitle("Cunt Movies by Rating Year")+ theme(axis.text.y =
element_text(size = rel(0.75)), axis.text.x =element_text(size =
rel(0.75)),legend.position="bottom")+

   facet_wrap(~ rating,ncol=2)
```



As we can see each year's rating is varied in the type of nostalgic to current films

2.5.1 Movie, User, Release Year, Rating Year and Nostalgia Level Effects Model

We now try to use out insights on the individual Rating year's biases to help fine tune our model that each user have a different seance of nostalgia than others. We represent this nostalgia level bias as b_n into our existing model.

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

where b_n is the bias for each level n.

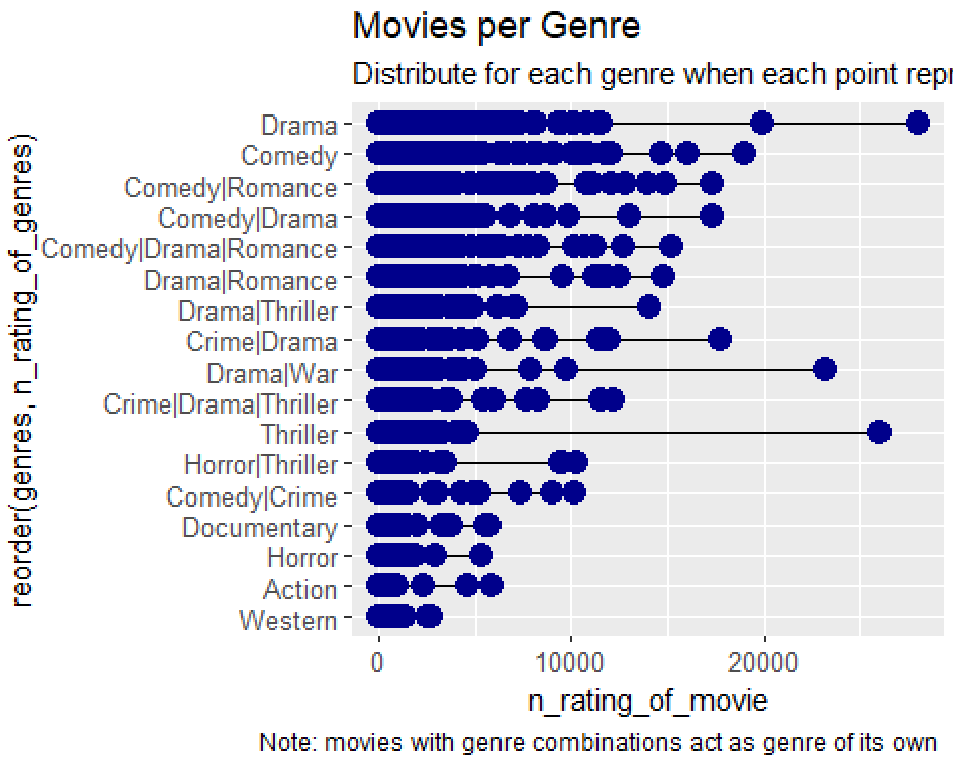
```
# Estimate nostalgia level  effect (b_n)
nostalgia_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(year_avgs, by = "releaseyear") %>%
  left_join(date_avgs, by = "ratingyear") %>%
  group_by(nostalgialevel) %>%
  summarise(b_n = mean(rating - mu_hat - b_i - b_u - b_y - b_r ))
# Predict ratings adjusting for movie, user, release year, rating date
and nostalgia effects
predicted_b_n <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(year_avgs, by = "releaseyear") %>%
  left_join(date_avgs, by = "ratingyear") %>%
  left_join(nostalgia_avgs, by = "nostalgialevel") %>%
  mutate(pred = mu_hat + b_i + b_u + b_y + b_r + b_n) %>%
  pull(pred)
 # Calculate RMSE based on review date effects model
 nostalgia_rmse <- RMSE(predicted_b_n, test_set$rating)

# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Movie, user, release year,
rating date and nostalgia level  Effects Model",
                          RMSE = nostalgia_rmse,
                          Prev_Diff=round(nostalgia_rmse-
ratingdate_rmse, 5)))
rmse_results %>% knitr::kable()
```

Method	RMSE	Prev_Diff
Naive Analysis by Mean	1.0611350	NA
Movie Effects Model	0.9441568	-0.11698
Movie & user Effects Model	0.8659736	-0.07818
Movie, user & release year Effects Model	0.8656023	-0.00037
Movie, user, release year & rating date Effects Model	0.8655016	-0.00010
Movie, user, release year, rating date and nostalgia level Effects Model	0.8652384	-0.00026

2.6 Predictor 6: Genres

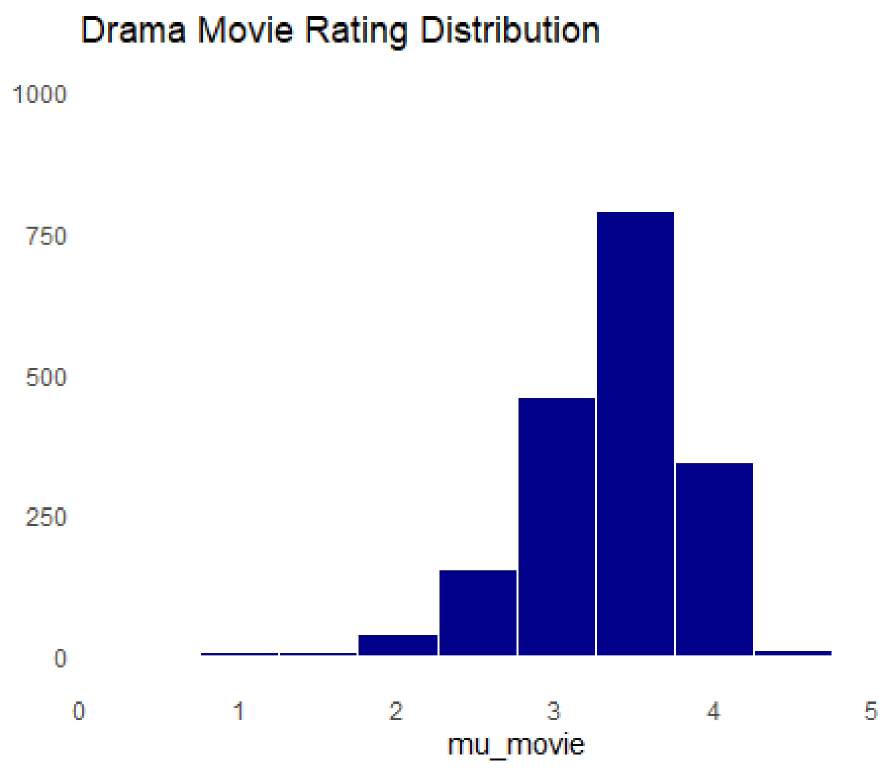
The last variable takes into account the movie genre and allows us to learn about their preferences. The amount of movies varies between the different genres as well as the amount of ratings each movie gets. There are movies that have been classified in a wider variety of genres and have been calculated as a genre of their own. Although this information can be used to make better predictions, this research doesn't use it. However it's worth exploring this information as well. As the data set contains 797 different combinations of genres.

```
 #genres

 genres_count_m<-edx1  %>%
   group_by(genres,movieId) %>%
   summarize(n_rating_of_movie = n(),
             mu_movie = mean(rating),
             sd_movie = sd(rating))

## `summarise()` has grouped output by 'genres'. You can override using
the
## `.groups` argument.

 genres_count_g<-edx1  %>%
   group_by(genres) %>%
   summarize(n_rating_of_genres = n())

 genres_count<-genres_count_m %>% left_join(genres_count_g,
by="genres")%>%
   mutate(rank=rank(n_rating_of_genres))%>%
   arrange(desc(n_rating_of_genres))


 genres_count %>%
   ggplot( aes(x=reorder(genres,n_rating_of_genres),
y=n_rating_of_movie)) +
   geom_segment( data = . %>% filter(rank>40), aes(xend=genres,
yend=0)) +
   geom_point(data = . %>% filter(rank>40), size=4,color="darkblue") +
   coord_flip() +
   ggtitle("Movies per Genre") +
   labs(fill = "Number of Ratings",
        subtitle="Distribute for each genre when each point represent a
movie ",
        caption ="Note: movies with genre combinations act as genre of
its own ") +
    theme(axis.text.x = element_text(size = 10),
                      axis.text.y = element_text(size = 10),
                      legend.position = "none")
```

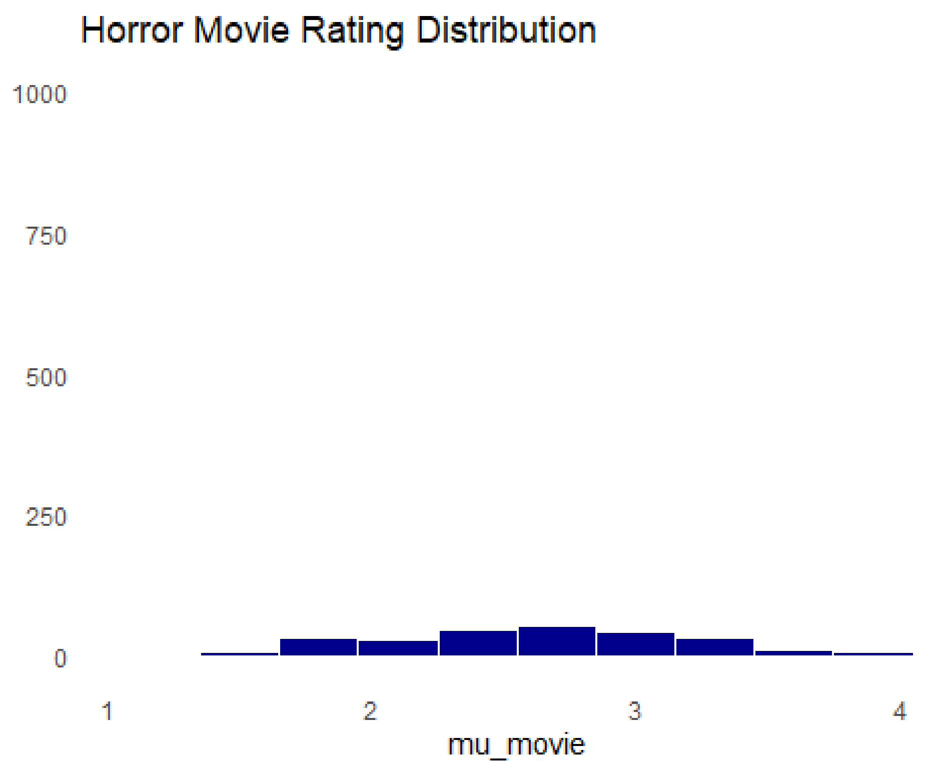


To confirm the necessity of the variable we will take an example of two genres. While Drama is in first place and rated the most times, Horror is far behind with fewer films of its type and fewer ratings. It follows that adding this variable will give us an important layer to our algorithm.

```
 # Drama rating distribution
 genres_count%>%filter(genres=="Drama") %>% ggplot(aes(x = mu_movie ))
+
   geom_histogram(bins=10,col="white", fill = "darkblue") +
   scale_y_continuous(limits = c(0, 1000))+
   ggtitle("Drama Movie Rating Distribution ") +
   theme_minimal()+ theme(panel.grid = element_blank(),axis.title.y =
element_blank())
```



```
 # Horror rating distribution
 genres_count%>%filter(genres=="Horror") %>% ggplot(aes(x = mu_movie ))
+
   geom_histogram(bins=10,col="white", fill = "darkblue") +
   scale_y_continuous(limits = c(0, 1000))+
   ggtitle("Horror Movie Rating Distribution ") +
   theme_minimal()+ theme(panel.grid = element_blank(),axis.title.y =
element_blank())
```



2.6.1 Movie, User, Release Year, Rating Year, Nostalgia Level and Genre Effects Model

We now try to use out insights on the individual genre biases to help fine tune our model acknowledging that genre affects all other previous predictors at different levels. We represent this genre bias as b_g into our existing model.

$$Y_{u,i} = \mu + b_i + b_u + b_y + b_r + b_n + b_g + \epsilon_{u,i}$$

where b_g is the bias for each user g.

```r
# Estimate genre effect (b_g)
genre_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(year_avgs, by = "releaseyear") %>%
  left_join(date_avgs, by = "ratingyear") %>%
  left_join(nostalgia_avgs, by = "nostalgialevel") %>%
  group_by(genres) %>%
  summarise(b_g = mean(rating - mu_hat - b_i - b_u - b_y - b_n))
# Predict ratings adjusting for movie, user and genre effects
predicted_b_g <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(year_avgs, by = "releaseyear") %>%
  left_join(date_avgs, by = "ratingyear") %>%
  left_join(nostalgia_avgs, by = "nostalgialevel") %>%
  left_join(genre_avgs, by = "genres") %>%
   mutate(pred = mu_hat + b_i + b_u + b_y  + b_n + b_g ) %>%
   pull(pred)
  # Calculate RMSE based on genre effects model
  genre_rmse <- RMSE(predicted_b_g, test_set$rating)

# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Movie, user, release year,
rating year, nostalgia and genre  Effects Model",
                          RMSE = genre_rmse,
                          Prev_Diff=round(genre_rmse-nostalgia_rmse,
5)))
rmse_results %>% knitr::kable()
```

Method	RMSE	Prev_Diff
Naive Analysis by Mean	1.0611350	NA
Movie Effects Model	0.9441568	-0.11698
Movie & user Effects Model	0.8659736	-0.07818
Movie, user & release year Effects Model	0.8656023	-0.00037
Movie, user, release year & rating date Effects Model	0.8655016	-0.00010
Movie, user, release year, rating date and nostalgia level Effects Model	0.8652384	-0.00026
Movie, user, release year, rating year, nostalgia and genre Effects Model	0.8649833	-0.00026

3 Regularized effect model

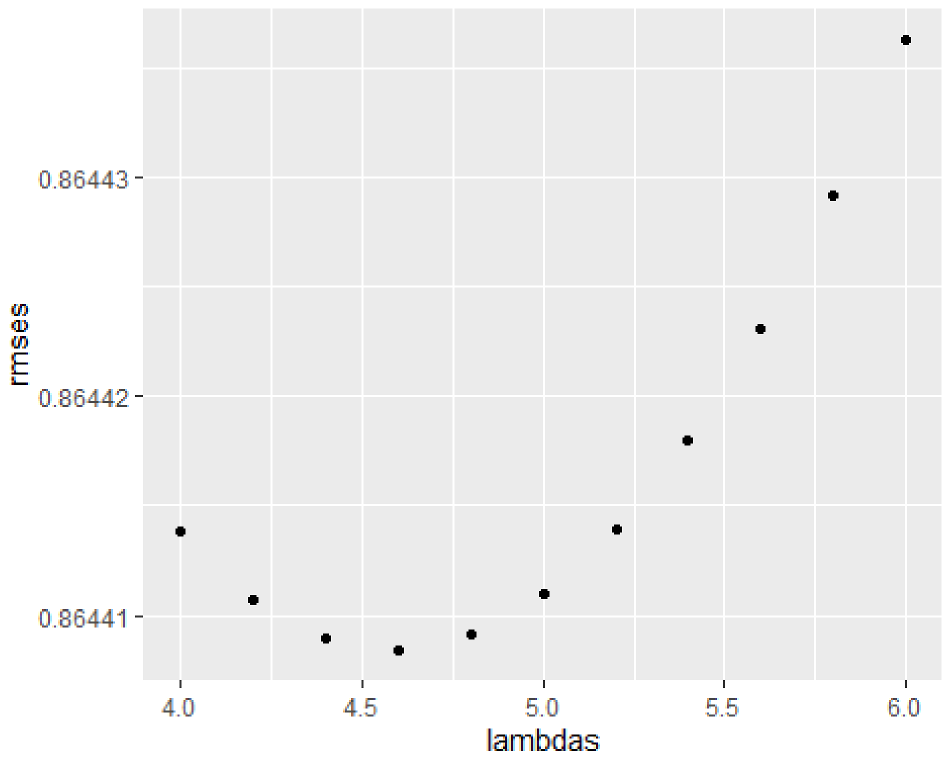
Regularization permits us to penalize large estimates that come from small sample sizes. It has commonalities with the Bayesian approach that shrunk predictions. The general idea is to add a penalty for large values of predictors to the sum of squares equation that we minimize. So having many large bi or bu makes it harder to minimize. A more accurate estimation of bu and bi will treat them symmetrically, by solving the least squares problem

$$Y_{u,i} = \mu + b_{i,n,\lambda} + b_{u,n,\lambda} + +b_{y,n,\lambda} + b_{r,n,\lambda} + b_{n,n,\lambda} + b_{g,n,\lambda} + \epsilon_{u,i}$$

lambda is a tuning parameter used for the penalty and cross-validation is used to choose from the optimal value. (for running purposes lets shrink the sequel to the minimum lambda result environment)

```
# Generate a sequence of values for lambda ranging from 4 to 6 with 0.2
increments (inc)
 lambdas <- seq(4, 6, 0.2)
 # Regularise model, predict ratings and calculate RMSE for each value
of lambda
 rmses <- sapply(lambdas, function(l){
   b_i <- train_set %>%
     group_by(movieId) %>%
     summarise(b_i = sum(rating - mu_hat)/(n()+l))
   b_u <- train_set %>%
     left_join(b_i, by="movieId") %>%
     group_by(userId) %>%
     summarise(b_u = sum(rating - b_i - mu_hat)/(n()+l))
   b_y <- train_set %>%
     left_join(b_i, by="movieId") %>%
     left_join(b_u, by="userId") %>%
     group_by(releaseyear) %>%
     summarise(b_y = sum(rating - b_i - b_u - mu_hat)/(n()+l))
   b_r <- train_set %>%
     left_join(movie_avgs, by = "movieId") %>%
     left_join(user_avgs, by = "userId") %>%
     left_join(year_avgs, by = "releaseyear") %>%
     group_by(ratingyear) %>%
     summarise(b_r = sum(rating - b_i - b_u - b_y - mu_hat)/(n()+l))
   b_n <- train_set %>%
     left_join(b_i, by="movieId") %>%
     left_join(b_u, by="userId") %>%
     left_join(b_y, by="releaseyear") %>%
     left_join(b_r, by="ratingyear") %>%
     group_by(nostalgialevel) %>%
     summarise(b_n = sum(rating - b_i - b_u - b_y - b_r -
mu_hat)/(n()+l))
   b_g <- train_set %>%
     left_join(b_i, by="movieId") %>%
     left_join(b_u, by="userId") %>%
     left_join(b_y, by="releaseyear") %>%
     left_join(b_r, by="ratingyear") %>%
     left_join(b_n, by="nostalgialevel") %>%
     group_by(genres) %>%
     summarise(b_g = sum(rating - b_i - b_u  - b_y - b_r - b_n -
mu_hat)/(n()+l))
   predicted_ratings <- test_set %>%
     left_join(b_i, by="movieId") %>%
     left_join(b_u, by="userId") %>%
     left_join(b_y, by="releaseyear") %>%
     left_join(b_r, by="ratingyear") %>%
     left_join(b_n, by="nostalgialevel") %>%
     left_join(b_g, by="genres") %>%
     mutate(pred = mu_hat + b_i + b_u + b_y + b_r + b_n + b_g) %>%
     pull(pred)
   return(RMSE(predicted_ratings, test_set$rating))
 })

 #plot lambdas
 qplot(lambdas, rmses)
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this
warning was
## generated.
```



```
 # Assign optimal tuning parameter (lambda)
 lambda <- lambdas[which.min(rmses)]
 lambda
```

```
## [1] 4.6
```

```
 # Minimum RMSE achieved
 regularised_rmse <- min(rmses)


# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Regularized predictors effect 
model",
                          RMSE = regularised_rmse,
                          Prev_Diff=round(regularised_rmse-genre_rmse, 
5)))
rmse_results %>% knitr::kable()
```

Method	RMSE	Prev_Diff
Naive Analysis by Mean	1.0611350	NA
Movie Effects Model	0.9441568	-0.11698
Movie & user Effects Model	0.8659736	-0.07818
Movie, user & release year Effects Model	0.8656023	-0.00037
Movie, user, release year & rating date Effects Model	0.8655016	-0.00010
Movie, user, release year, rating date and nostalgia level Effects Model	0.8652384	-0.00026
Movie, user, release year, rating year, nostalgia and genre Effects Model	0.8649833	-0.00026
Regularized predictors effect model	0.8644084	-0.00057

4 Validating the preferred model using the final_holdout_set.

4.1 results

Now we use our preferred Model on the final_holdout_test set to validate the success of our findings.

```
 # Use mutate function to update validation dataset in line with changes made to edx
 final_holdout_test1 <- final_holdout_test%>%
mutate(ratingdate=as_date(as_datetime(timestamp)),

ratingyear=as.numeric(format(ratingdate,"%Y")),

ratingmonth=as.numeric(format(ratingdate,"%m")),
                                                releaseyear =
as.numeric(str_extract(str_extract(title, "[/(]\\d{4}[/)]$"),
regex("\\d{4}")),title = str_remove(title, "[/(]\\d{4}[/)]$")),

nostalgialevel=ratingyear-releaseyear
                                                     )
```

A machine learning algorithm to predict the ratings from the Movie Lens data set was constructed. The optimal model incorporated the effects of various bias predictor variables and were regularized in the model. We compare validation RMSE versus requirement RMSE to give some context to the results.

```
# Use full edx dataset to model all effects + regularised with chosen value for lambda
 b_i <- edx1 %>%
   group_by(movieId) %>%
   summarise(b_i = sum(rating - mu_hat)/(n()+lambda))

 b_u <- edx1 %>%
   left_join(b_i, by="movieId") %>%
   group_by(userId) %>%
   summarise(b_u = sum(rating - b_i - mu_hat)/(n()+lambda))

 b_y <- edx1 %>%
   left_join(b_i, by="movieId") %>%
   left_join(b_u, by="userId") %>%
   group_by(releaseyear) %>%
   summarise(b_y = sum(rating - b_i - b_u - mu_hat)/(n()+lambda))

 b_r <- edx1 %>%
   left_join(movie_avgs, by = "movieId") %>%
   left_join(user_avgs, by = "userId") %>%
   left_join(year_avgs, by = "releaseyear") %>%
   group_by(ratingyear) %>%
   summarise(b_r = sum(rating - b_i - b_u - b_y - mu_hat)/(n()+lambda))

 b_n <- edx1 %>%
   left_join(b_i, by="movieId") %>%
   left_join(b_u, by="userId") %>%
   left_join(b_y, by="releaseyear") %>%
   left_join(b_r, by="ratingyear") %>%
   group_by(nostalgialevel) %>%
   summarise(b_n = sum(rating - b_i - b_u - b_y - b_r -
mu_hat)/(n()+lambda))

 b_g <- edx1 %>%
   left_join(b_i, by="movieId") %>%
   left_join(b_u, by="userId") %>%
   left_join(b_y, by="releaseyear") %>%
   left_join(b_r, by="ratingyear") %>%
   left_join(b_n, by="nostalgialevel") %>%
   group_by(genres) %>%
   summarise(b_g = sum(rating - b_i - b_u  - b_y - b_r - b_n -
mu_hat)/(n()+lambda))

  predicted_ratings <- final_holdout_test1 %>%
   left_join(b_i, by="movieId") %>%
   left_join(b_u, by="userId") %>%
   left_join(b_y, by="releaseyear") %>%
   left_join(b_r, by="ratingyear") %>%
   left_join(b_n, by="nostalgialevel") %>%
   left_join(b_g, by="genres") %>%
   mutate(pred = mu_hat + b_i + b_u + b_y + b_r + b_n + b_g) %>%
   pull(pred)

   # Calculate final validation RMSE
 valid_rmse <- RMSE(final_holdout_test1$rating, predicted_ratings)

 rmse_requirement=0.86490

final_rmse_results = tibble(Method = "RMSE edx requirement", RMSE =
rmse_requirement)

 # Save Results to the Table
final_rmse_results <- bind_rows(final_rmse_results,
                          tibble(Method="Validation of Final Model",
                                 RMSE = valid_rmse,
                                 Prev_Diff=round(valid_rmse-
rmse_requirement, 5)))
final_rmse_results %>% knitr::kable()
```

Method	RMSE	Prev_Diff
RMSE edx requirement	0.8649000	NA
Validation of Final Model	0.8638161	-0.00108

The goal of the project was to develop an algorithm with an RMSE obtained from it lower than 0.8649

4.2 conclusion

As we added layers of explanatory variables that managed to decrease the RMSE. Each variable reduces the distance between the rating obtained from the regression model (predicted values) and the rating from the test sample (observed values). Although both predictors movieId and userId accounted for the largest decrease. Still, each variable managed to reduce the error distance slightly more and contributing to the accuracy which provided a satisfactory result and more efficient algorithm.