EdX Movielens Project

Ayomide Bamgbose

23/12/2020

Executive Summary

This EdX project defines a linear regression machine learning model that will predict the rating of a given movie. The model will be created with the 10M MovieLens dataset. Due to the size of this dataset, an approximation of a linear effects model will be used to reduce the algorithm run-time. The `userId` and `movieId` predictors consistently had the highest impact on the RMSE of the model. The selected predictors for the linear regression model are `userId`, `movieId`, `movie_year` and `genres`. The RMSE of the selected model is **0.86475** when tested on the `edx_test` dataset, and **0.86516** when tested on the `validation` dataset. The equation of the linear effects model is $Y = \mu + x_u + x_i + x_g + x_{my}$, and a tuning parameter $\lambda = 4.5$ is used to regularize each predictor. The main limitation of this project is the usage of an approximation of the linear model.

Introduction

The goal of this EdX project is to create a movie recommendation system for the 10M version of the MovieLens dataset. An algorithm will be developed with this dataset after it has been cleaned, explored, and visualized. This algorithm will predict the rating a movie will receive; the minimum possible rating is 0/5 and the maximum possible rating is 5/5. Given the large amount of data in the MovieLens dataset, an approximation of the linear effects model (i.e., linear regression `lm()`) will be used to create the final algorithm. The root mean-squared error (RMSE) of the model will also be determined to validate the accuracy of the selected machine learning model.

Overview

Before an algorithm can be created, the 10M MovieLens dataset must first be cleaned. After the data has been cleaned, a training set (`edx_train`), test set (`edx_test`), and validation set (`validation`) will be created:

- **`edx_train`**: this data will be used to train the machine learning models
- **`edx_test`**: this data will be used to perform tests (and determine the RMSE) on the machine learning models
- **`validation`**: this data will be used to determine the final RMSE value with the selected machine learning model

This model will approximate the linear effects machine learning model (`lm()`), where Y is the rating, μ is the average rating, ϵ is the error, and x_1 to x_n are the effects of predictors 1 to n: $Y = \mu + x_1 + x_2 + ... + x_n + \epsilon$. The 10M MovieLens dataset contains the following variables:

- **`userId`**: the ID of the user giving the movie rating
- **`movieId`**: the ID of the movie being rated
- **`rating`**: the rating given by the user
- **`timestamp`**: the time that the rating was given
- **`title`**: the title of the movie being rated
- **`genres`**: the genre(s) of the movie being rated

The characteristics and relationships between each of these variables will be determined and visualized in the *Data Visualization* section. After this, the effects and variability of each variable will be determined. If necessary, certain variables will be regularized before they are used in the prediction model. An additional variable (`movie_year`) will be added to the dataset (see *Data Cleaning*).

A variety of variable-predictor combinations will be used for the linear model. The model with the lowest RMSE will be selected as the final model and will be used to determine the RMSE of the final hold-out test set, `validation`. The RMSE will be calculated with the following equation:

$RMSE = \sqrt{\sum (ratings_{true} - ratings_{predicted})/n}$

The confidence interval CI is defined as: $CI = \mu + -2 * SE$, where μ is the average and SE is the standard error ($SE = s/\sqrt{n}$). In this case s is the standard deviation and n is the number of samples.

Note: The following R packages will be used in this project: `tidyverse`, `caret`, `lubridate`, `gridExtra`, and `data.table`. The `extrafont` package will be used to format the plots.

Data Cleaning

Since the data for this project is being imported from an external source, it must be converted into a manageable data structure so that it can be used in R. The code for this process is provided below:

```r
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(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

The final dataset is the `movielens` dataset. An additional predictor `movie_year` will be added to this data set with the `movie_to_year()` function (below). This function will extract the year of release of a movie by taking a substring of each row of the `title` column. Since each year is four digits long and is always at the end of the `title` string, it can be extracted as the last four digits of each movie `title` (ignoring the last end bracket ")"):

```r
movie_to_year <- function(movie){
  temp <- substr(movie, nchar(movie)-4, nchar(movie)-1)
  as.numeric(temp)
}

# Example:
movie_to_year("Star Trek: Generations (1994)")
```

```
## [1] 1994
```

From this dataset, the `edx` and `validation` datasets will be formed. The `validation` set will be composed of 10% of the `movielens` dataset. The `edx_test` and `edx_train` datasets will be created from the `edx` dataset, and the `edx_test` set will contain 20% of the `edx` dataset. These steps are conducted after setting the seed to **1** (to ensure that the response is consistent).

The variables (i.e., predictors) in the `edx_train`, `edx_test` and `validation` sets are:

```
## Classes 'data.table' and 'data.frame':   7200089 obs. of  7 variables:
##  $ userId    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId   : num  185 316 329 355 364 377 420 539 588 589 ...
##  $ rating    : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp : int  838983525 838983392 838983392 838984474 838983707 838983834 838983834 838984068 8
##  $ title     : chr  "Net, The (1995)" "Stargate (1994)" "Star Trek: Generations (1994)" "Flintstones
##  $ genres    : chr  "Action|Crime|Thriller" "Action|Adventure|Sci-Fi" "Action|Adventure|Drama|Sci-Fi
##  $ movie_year: num  1995 1994 1994 1994 1994 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Methods

Although the `movielens` dataframe has been split into a training set (`edx_train`) and two testing sets (`edx_test` and `validation`), each dataset has a large amount of data (see *Data Exploration*). The linear effects (i.e., linear regression) model is usually implemented with the `lm()` function. If we were to use this function, the run-time would be too high due to the size of the dataset and number of unique entries in each predictor (see *Data Exploration*). Due to this, an estimation of this linear model will be used instead: $Y = \mu + x_u + x_m + x_g + x_t + x_{my} + \epsilon$, where Y is the predicted rating, μ is the average rating, x_u is the user (`userId`) effect, x_m is the movie (`movieId`, `title`) effect, x_g is the genre (`genres`) effect, x_t is the time (`timestamp`) effect, x_{my} is the year (`movie_year`) effect and ϵ is the error. The predictors that will be used in the final model will be determined by considering their effect on the movie rating and RMSE of the prediction model.

Data Exploration

The dimension of the `validation`, `edx_test` and `edx_train` sets are defined as:

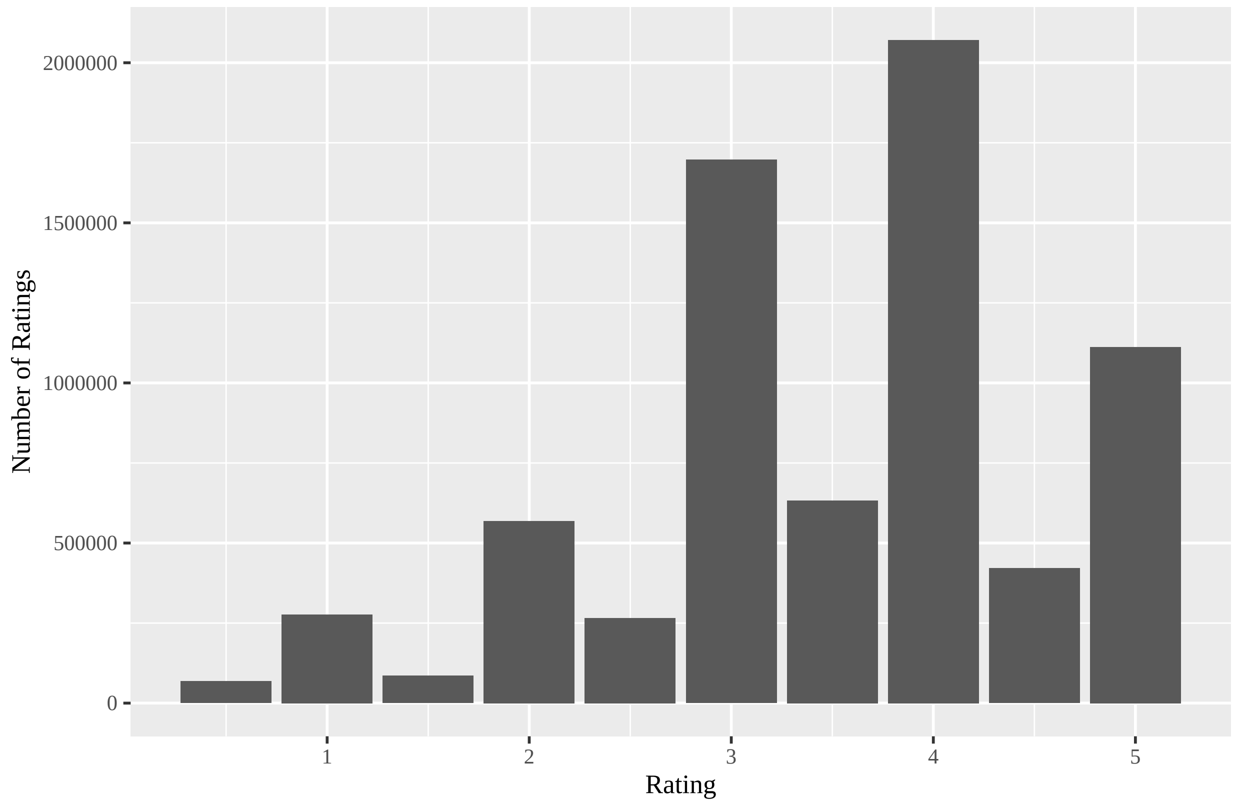
```
##     data_set row_dimension col_dimension
## 1 validation        999999             7
## 2   edx_test       1799966             7
## 3  edx_train       7200089             7
```

Further analysis will be conducted on the `edx_train` set only because this dataset will be used to create and train the machine learning model.

Rating Distribution

The rating distribution of the `edx_train` set is given in Figure 1 below:

Figure 1. Rating Distribution (in the training set)



From this figure, it is evident that the top 3 most common ratings are 4/5, 3/5, and 5/5, where 28.8% of all ratings are 4/5, 23.6% of all ratings are 3/5, and 15.5% of all ratings are 5/5. The `rating` variable is not normally distributed.

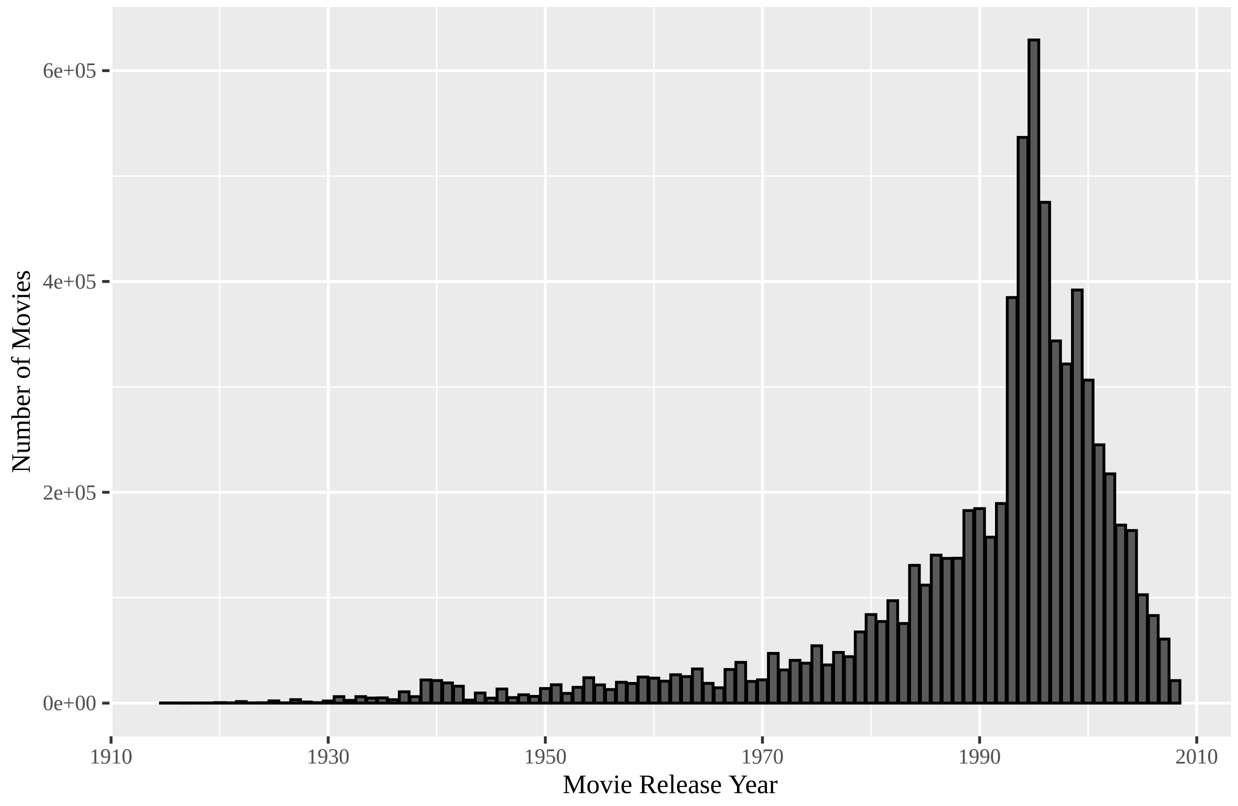
Genres, Users, and Movies

The number of distinct users (`userId`), movies (`movieId`), and genres (`genres`) in the `edx_train` dataset are given in the table before:

```
##   n_movies n_users n_genres
## 1    10677   69878      797
```

From Figure 2, below, it follows that the majority of the movies in the `edx_train` dataset were released between 1990 and 2010:

Figure 2. Number of Movies Released vs Year of Release



Data Visualization

This section will be conducted on the `edx_train` dataset, and will be used to:

- Learn more about the behaviour of each predictor
- Find key trends and identify the relationship between each predictor and the movie rating

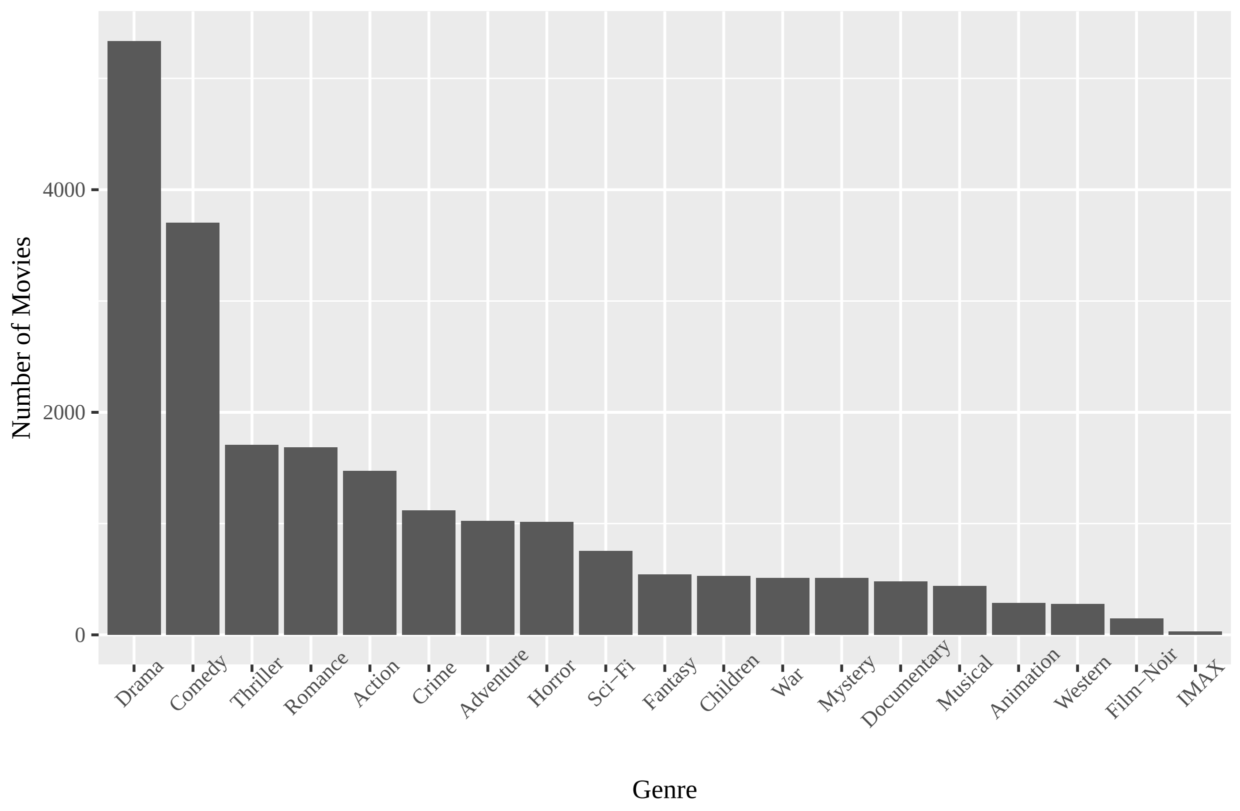
Genres (`genres`)

The `genres` predictor is composed of combinations of the unique genres in the `edx_train` dataset. Some movies have multiple genres, and some movies have only one genre. For example:

```
##                   title                genres
## 1:     Net, The (1995) Action|Crime|Thriller
## 2: Multiplicity (1996)                Comedy
```

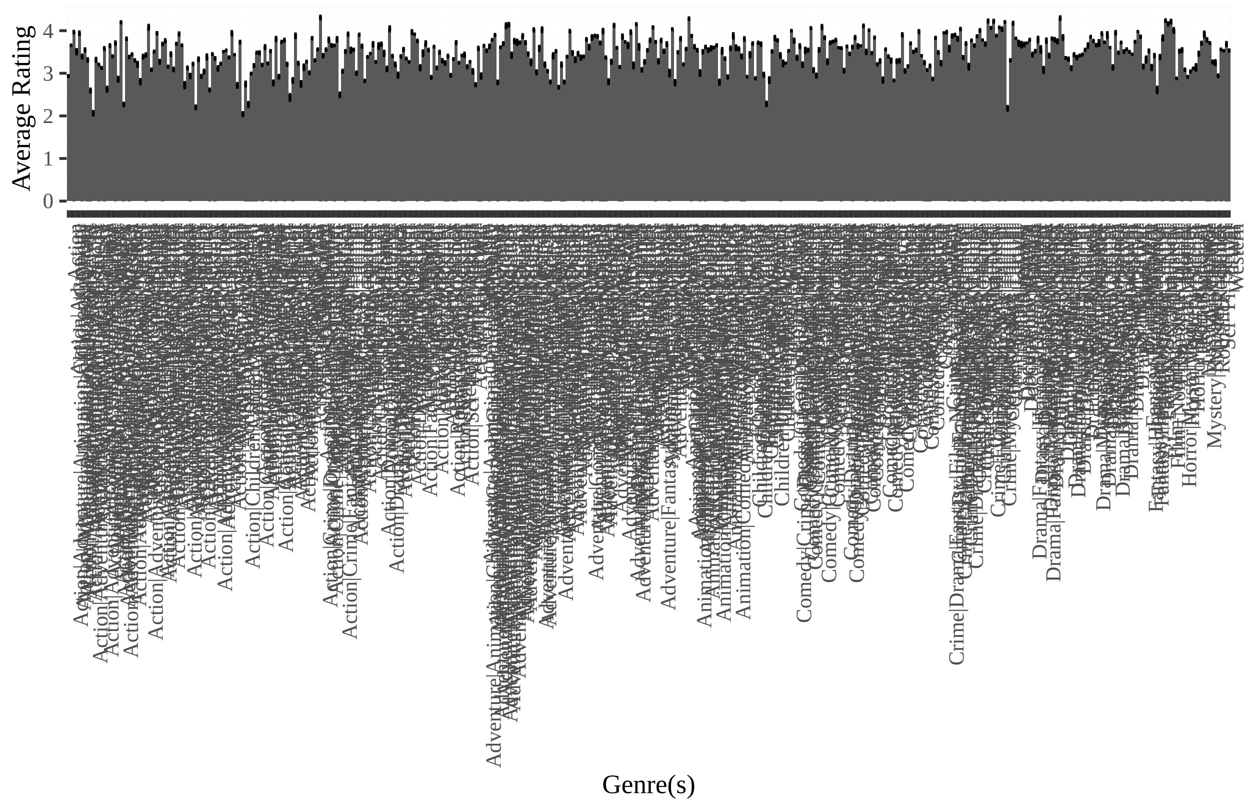
There are 19 unique genres in the `edx_train` dataset, and the number of movies per genre are given in Figure 3, below:

Figure 3. Number of Movies per Genre



The most common genres in the `edx_train` dataset are `Drama`, `Comedy`, and `Thriller`. The least common genres are `Western`, `IMAX`, and `Film-Noir`. Although there are 19 unique genres, there are actually `797` unique combinations of genres in the `edx_train` dataset. Considering all the genre combinations with more than 1000 ratings, the following distribution is observed:

Figure 4. Rating Distribution per Genre



Since there are so many genre combinations, this plot is very difficult to interpret. By counting the number of genres per movie (using only the **unique** movies in the dataset), the following can be shown:

```
## n_genres
##    1    2    3    4    5    6    7    8
## 4004 3701 2004  744  186   34    3    1
```

In other words, by considering only the first **three** genres of each movie, we can account for `90.9%` of all movies in the `edx_train` dataset (i.e., `90.9%` of all movies in the `edx_train` dataset have 1, 2, or 3 distinct genres).

Three "predictors" can now be added to the `edx_train` dataframe: `genre_1`, `genre_2` and `genre_3`. By separating the *genres* column along each "`|`", the first three genres of each movie can be determined. In the case that a movie does not have up to three genres, the empty values are filled with the string `"(no genre listed)"`. Now, `edx_train` is defined as:

```
## Classes 'data.table' and 'data.frame':   7200089 obs. of  9 variables:
##  $ userId    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId   : num  185 316 329 355 364 377 420 539 588 589 ...
##  $ rating    : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp : int  838983525 838983392 838983392 838984474 838983707 838983834 838983834 838984068 8
##  $ title     : chr  "Net, The (1995)" "Stargate (1994)" "Star Trek: Generations (1994)" "Flintstones
##  $ genre_1   : chr  "Action" "Action" "Action" "Children" ...
##  $ genre_2   : chr  "Crime" "Adventure" "Adventure" "Comedy" ...
##  $ genre_3   : chr  "Thriller" "Sci-Fi" "Drama" "Fantasy" ...
##  $ movie_year: num  1995 1994 1994 1994 1994 ...
```

In other words, the effect x_g is now defined as: $x_g = x_{g1} + x_{g2} + x_{g3}$, where x_{g1}, x_{g2} and x_{g3} are the effects for `genre_1`, `genre_2`, and `genre_3`, respectively.

Genre 1, 2, and 3 As shown in Figure 5 below, each genre has its own average rating. Excluding the movies with genre "`(no genres listed)`" or "`IMAX`", all genres have very low variability. The "`(no genres listed)`" are "`IMAX`" the genres with the lowest number of ratings, thus they will cause a lot of noise in the data. Ignoring these genres, the confidence intervals of all other genres in this dataset do not overlap. This suggests that the first genre of a movie has a statistically significant effect on its rating.

Figure 5. Ratings, Genre 1

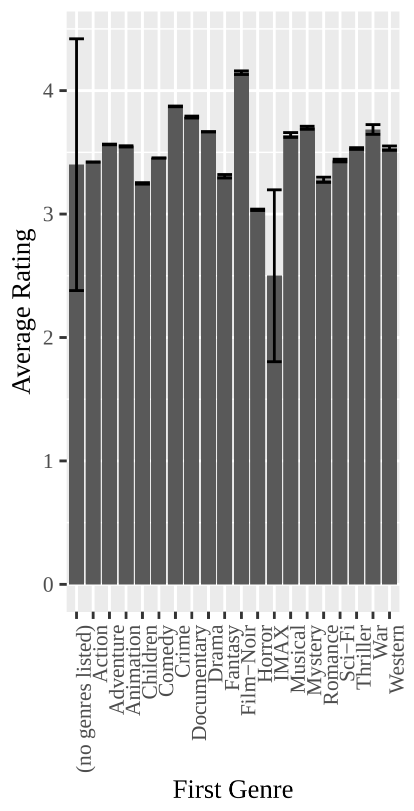


Figure 6. Rating, Genre 2

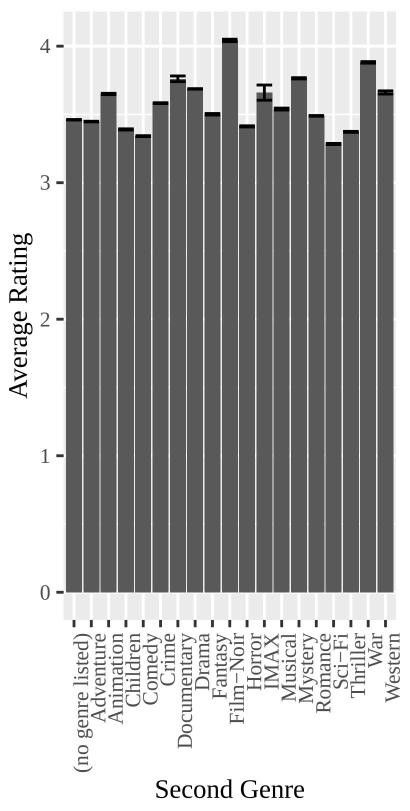
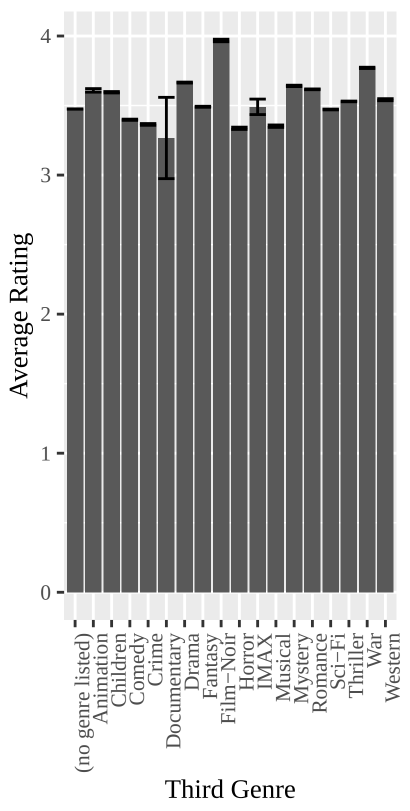


Figure 7. Ratings, Genre 3



In a similar manner to the plot for the first genre, Figure 6 shows that the confidence intervals for each genre in `genre_2` do not overlap, indicating that the second genre of a movie has a statistically significant effect on its average rating.* The genre with the highest variability in this plot is the "`IMAX`" genre.

In a similar manner to the first and second genres, Figure 7 shows that the confidence intervals for each genre do not overlap, indicating that the third genre of a movie has a statistically significant effect on its average rating.* The genres with the highest variability in this case are the "`Documentary`" and "`IMAX`" genres.

*These statements will be confirmed in the *Prediction Models* section of the report.

Users (`userId`)

There are 69,878 unique users in the `edx_train` dataset. The distribution of the number of ratings given by users, filtered to include only users with more than 500 total ratings, is provided in Figure 9 below. From this plot, it is evident that almost 100% of the users in the `edx_train` set have given less than 2000 total movie ratings.

The average movie rating given per user is presented in Figure 8. From this figure, it is evident that the most common (rounded) rating is 3.5/5, and the least common ratings are at the edges of the rating scale (i.e., 0-2/5 or 5/5). 42.6% of the average user ratings are 3.5/5, and less than 1% of the average user ratings are less than 2/5 or equal to 5/5.

Figure 9 represents the relationship between the average rating given to a movie (by a unique user) and the number of ratings that user has given. From this figure, it is evident that the more ratings a user gives, the more "average" their ratings are: "average", in this case, meaning between 2/5 and 4/5. However, when a user has rated fewer movies, their ratings tend to be either very high (i.e., >4/5) or very low (i.e., <2/5). The impact of this observation will be considered in more detail in the *Predictors:Effects* section of this report.

Figure 8. Number of Ratings per User

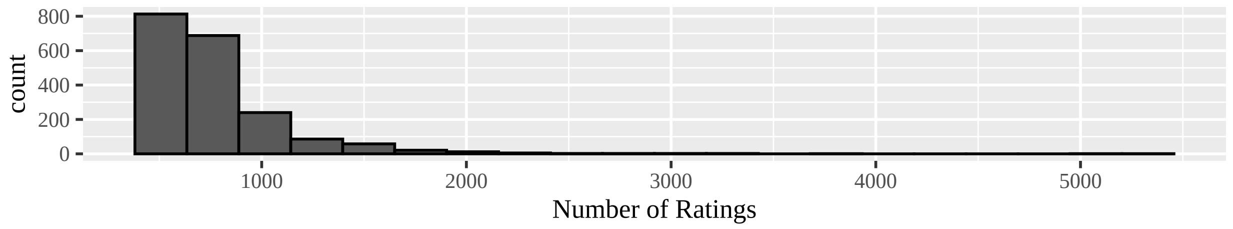


Figure 9. Number of Users versus Average Movie Rating

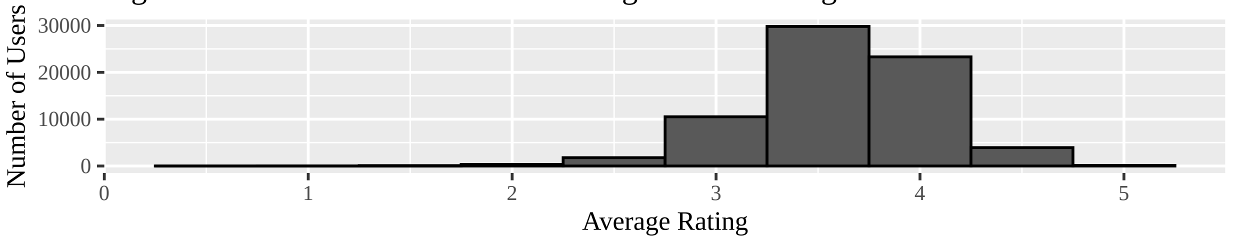


Figure 10. Average Rating versus number of Ratings per User

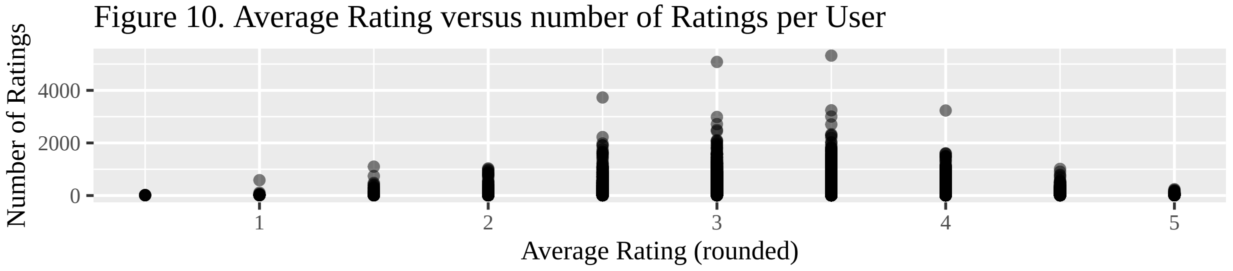


Figure 10 shows a similar distribution to Figure 8: the more ratings a user gives, the more "average" (between 3/5 and 4/5) their rating is. Users with a very low number of ratings tend to give either very high or very low ratings (i.e., less than 2/5 or 5/5). Since the `userId` effect already has some variability (as shown in the figures above), it may need to be regularized (i.e., penalized least squares) in the linear model.

Note: Each rating average is rounded to the nearest 0.5 by multiplying the average rating by two, rounding it with the `round()` function in R, and dividing the result by two. This data transformation is conducted so that the plots in Figure 8 and 9 are easier to create.

Movies (`movieId`, `title`, `movie_year`)

In the same way as the `userId` variable:

Figure 11. Number of Ratings per Movie

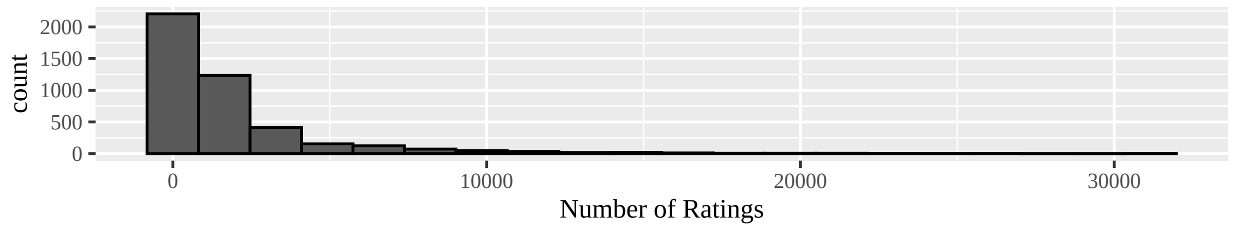


Figure 12. Number of Movies versus Average Movie Rating

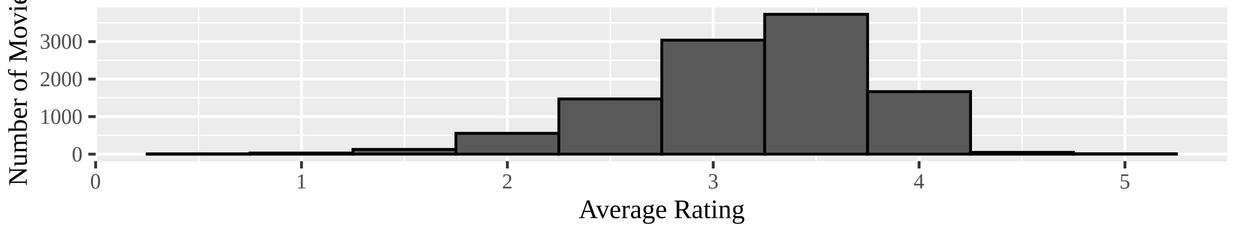


Figure 13. Average Rating vs Number of Ratings per Movie

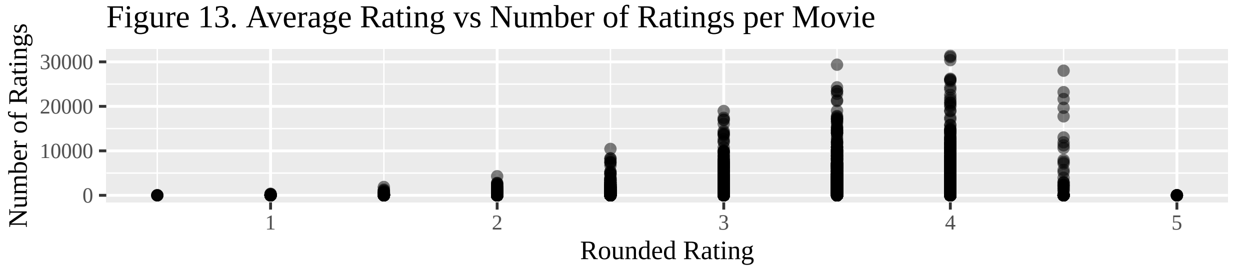
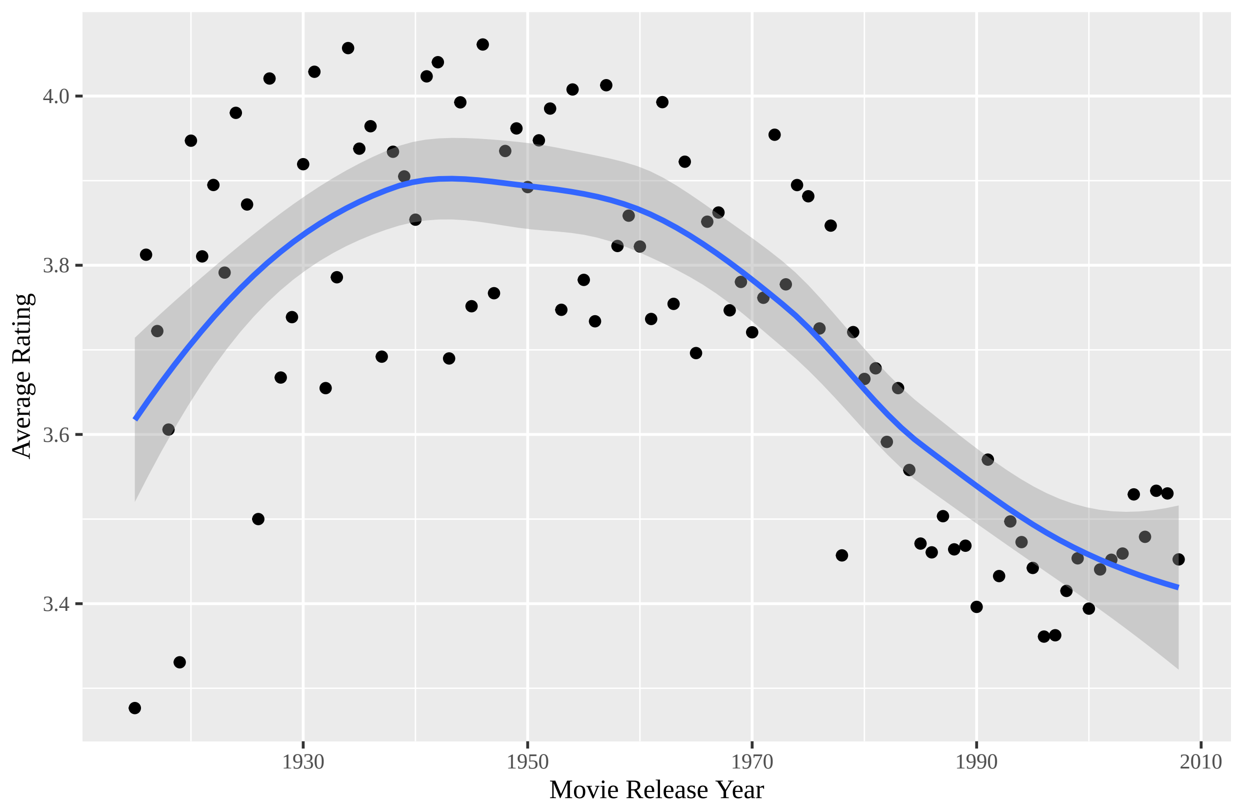


Figure 11 shows that almost 100% of all movies have less than 10,000 total ratings. By Figure 12, it follows that the most common movie rating is 3.5/5. Figure 13 shows that the more ratings a movie has, the more "average" its rating is (i.e., between 3/5 and 4/5). However, movies with very few ratings tend to have extremely low or high ratings (i.e., <2/5 or 5/5). The `movieId` effect will likely cause variability in the linear model: regularization (i.e., penalized least squares) may be required to deal with the extreme cases.

Figure 14, below, shows that there is a clear relationship between the release year of a movie (`movie_year`) and its rating. Movies released between 1910 and 1970 tend to have higher average ratings (ranging from 3.6/5 to 5/5). The movies released after 1970 tend to have a lower rating (<3.6/5), and show a trend of consistent decrease in the average rating:

Figure 14. Average Rating vs Movie Release Year

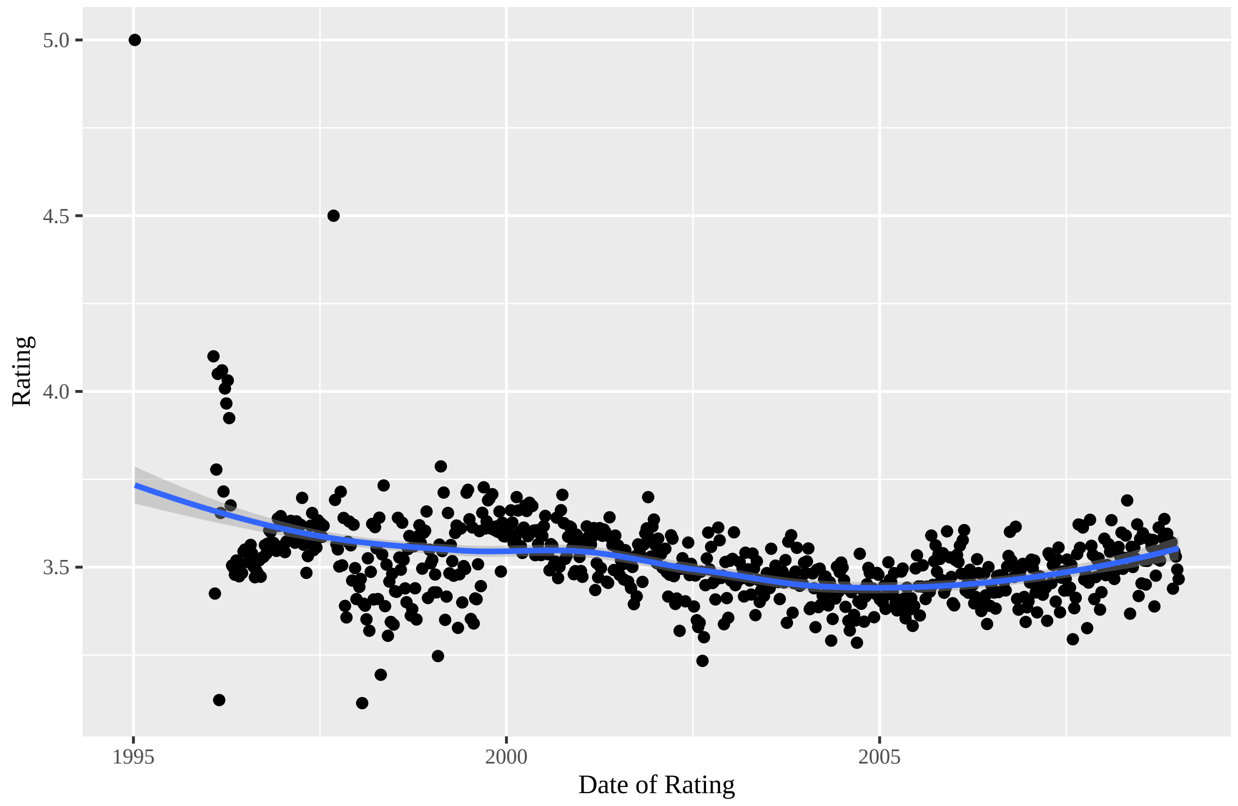


The trendline of the average rating versus movie release year plot (Figure 14) is created with the `geom_smooth()` function and `lm()` formula.

Time (timestamp)

The relationship between the time a rating was made (`timestamp`) and the given rating is provided in the figure below. From this plot, it is evident that the time predictor x_d can be described in terms of a non-linear and time-dependent function. In other words: $x_d \to f(x_d)$, where $f(x_d)$ is a smooth function of x_d.

Figure 15. Date of Rating versus Rating Given



This plot has been fitted with the `loess()` function. Considering the fitted blue `geom_smooth()` function, there is very little variation in the projected rating in this plot. It is likely that this predictor will not have much of an effect on the final rating given to a movie, but this will be determined in the *Predictors: Effects* section of the report.

Analysis

Considering the visualizations conducted in the previous section, the effects and impacts of each predictor in the `edx_train` will be analyzed in terms of the linear effects model: $Y = \mu + x_u + x_m + x_g + x_t + x_{my} + \epsilon$, where $x_g = x_{g1} + x_{g2} + x_{g3}$. Before the effects of each predictor can be analyzed, the average rating across all predictors (μ) is given by:

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

```
## [1] 3.5125
```

Predictors

Effects

For the purpose of this report, “effects” will be defined as the deviation of the rating from μ. For example, for the `userId` predictor, its effect (x_u) is defined as: $x_u = Y - \mu$, where this formula is defined for a specific rating Y. Since an approximation of the linear effects model will be used, this effect x_u will be the *average* effect of all ratings given by a specific user.

Figures 16 to 19 show the `userId` effect, `movieId` effect, `movie_year` effect, and `timestamp` effect, respectively. The `userId` effect is defined as $x_u = Y - \mu$ (grouped by unique users), `movieId` effect is defined as $x_i = Y - \mu$ (grouped by unique movies), `movie_year` effect is defined as $x_{my} = Y - \mu$ (grouped by release year), `timestamp` effect is defined as $x_d = Y - \mu$ (grouped by unique weeks).

Figure 16. User Effect, b_u

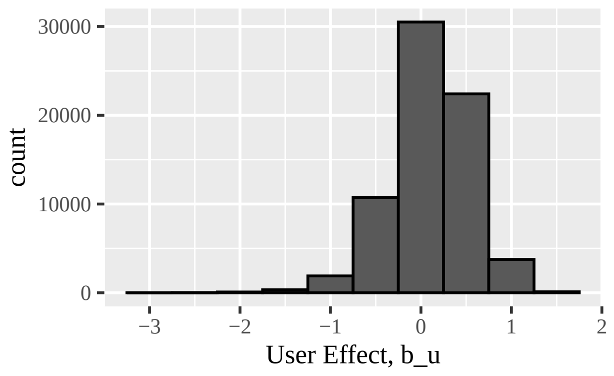


Figure 17. Movie Effect, b_i

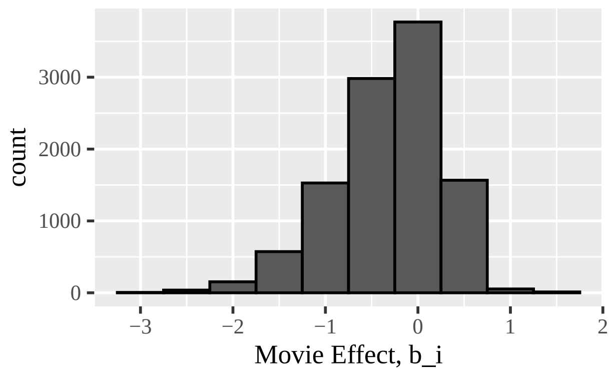


Figure 18. Date Effect, d_ui

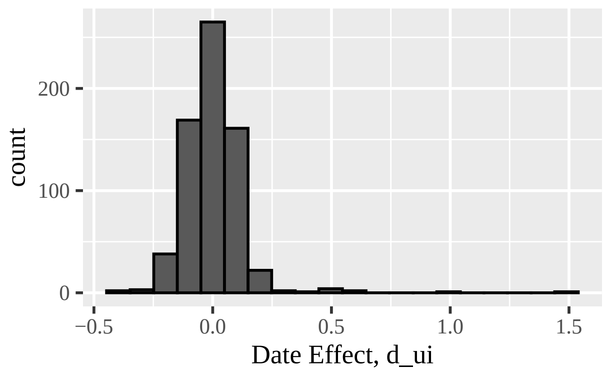
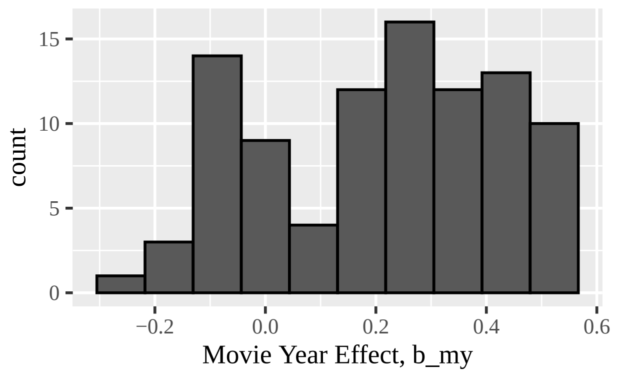


Figure 19. Movie Year Effect, b_my



From these plots, it is evident that all of these predictors have an effect on the rating a given movie will receive. The effects of the `userId` and `movieId` predictors are the most profound: they range from -3 to 2. The effect of the `timestamp` predictor, however, only ranges from -0.25 to 0.25 for the majority of the dataset, confirming that the `timestamp` predictor has a very low effect on the rating a movie receives. The effect of the `timestamp` predictor is only >0.25 in extreme cases (i.e., these effects will have very high variability). The effect of the `movie_year` predictor varies from -0.2 to 0.6.

The effects of the `genres`, `genre_1`, `genre_2`, and `genre_3` predictors are given in the figures below. The `genres` effects is defined as $x_g = Y - \mu$ (grouped by unique genres), `genre_1` effects is defined as $x_{g1} = Y - \mu$ (grouped by unique genre_1), `genre_2` effects is defined as $x_{g2} = Y - \mu$ (grouped by unique genre_2), and `genre_3` effects is defined as $x_{g3} = Y - \mu$ (grouped by unique genre_3).

Figure 20. Genre Effect

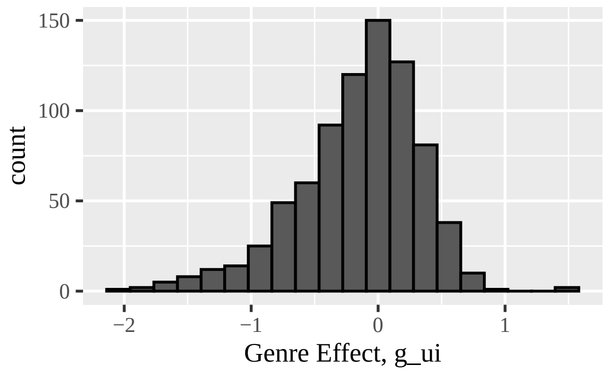


Figure 21. First Genre Effect

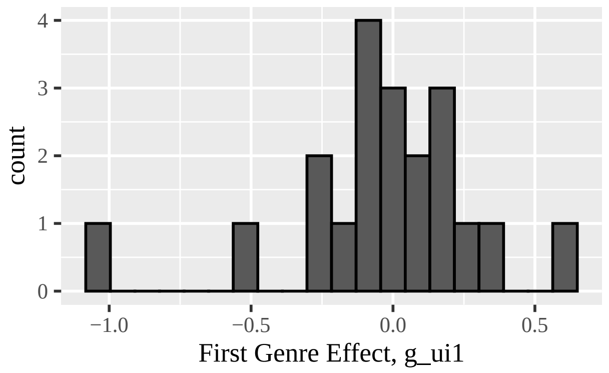


Figure 22. Second Genre Effect

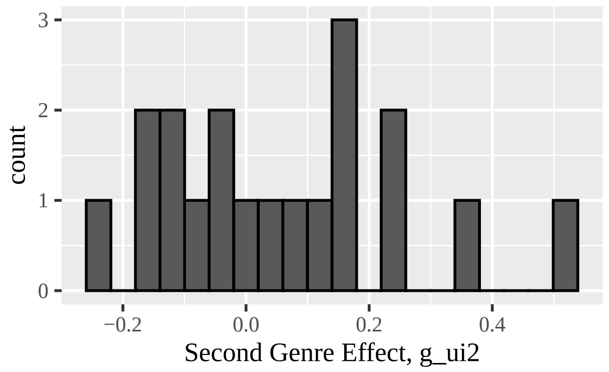
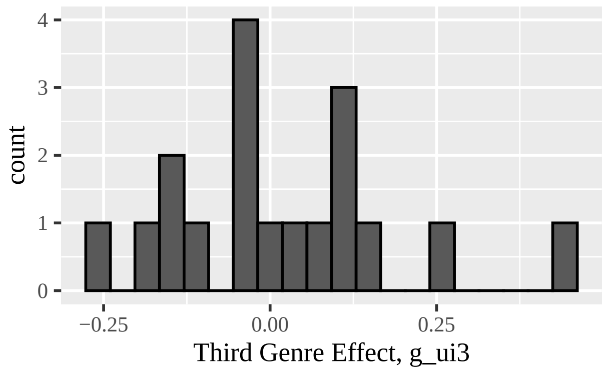


Figure 23. Third Genre Effect



From these plots, it is evident that each all of the genre predictors have an effect on the rating given to a movie. The `genres` predictor has the greatest effect on the rating of a movie: its effect ranges from -2 to 1.5. The `genre_1` and `genre_2` effects also have significant effects, but the `genre_3` has a very low effect on the rating (it ranges from -0.25 to 0.25). In other words, the `genres`, `genre_1` and `genre_2` have the greatest effect on the rating given to a movie.

Variability: Is Regularization Required?

In order to reduce the effect of variability, penalized least squares (regularization) will be tested for the following variables:

- `userId`: There is a significant variation in the `userId` effect on a movie's rating (Figure 16)
- `movieId`: There is a significant variation in the `movieId` effect on a movie's rating (Figure 17)
- `movie_year`: There is a significant variation in the `movie_year` effect on a movie's rating (Figure 18)
- `genre_1`: The `(no genre listed)` and `IMAX` genres have high variability (Figure 21)

In addition, the highest and lowest ratings are given by users with the less than 150 total ratings. However, the range of ratings given by users is:

```
range(unique_users$n)
```

```
## [1]    7 5323
```

In the same way, the highest and lowest ratings are given to movies with less than 200 total ratings. However, the range of ratings given to movies is:

```
range(unique_movies$n)
```

```
## [1]     1 31362
```

Therefore the `userId` and `movieId` predictors both have noisy estimates. This noise will increase the RMSE of the linear model, therefore regularization will be required to reduce the "effect" of these extreme cases (i.e., very high or very low ratings).

Penalized least squares will be implemented with the penalty term λ. Instead of calculating the estimates as the average of $x_i = Y - \mu$ (for some predictor i), they will be calculated with the following equation:

$x_i = (1/(\lambda + n_i)) \sum(Y - \mu)$.

The value of λ will be selected through cross-validation (see *Regularized Models* section).

Results

Prediction Models

The types of linear effects models will be considered in this report:

1. Naive ($Y = \mu$)
2. Non-regularized
3. Regularized (i.e., penalized least squares)

The root mean squared error (RMSE) will be used to determine the effectiveness of the intermediate models. The selected models will be used to estimate the ratings of the data in the `edx_test` dataset. As soon as a final model has been selected, the RMSE will be calculated by running the model on the `validation` dataset. The RMSE equation is defined as: $RMSE = \sqrt{\sum(ratings_{true} - ratings_{predicted})/n}$.

Naive Model

This model will perform predictions with the following linear equation: $Y = \mu$. The root mean squared error of this model is:

```
## Naive Model (no predictors used)
naive_model_RMSE <- RMSE(edx_test$rating, mu)
naive_model_RMSE
```

```
## [1] 1.0599
```

This model will be used as a baseline to measure the improvement of models with additional predictors.

Un-Regularized Models

Single Effect Models Single effect models apply the rating average and **one** additional predictor/effect to the linear model. Considering the general linear model: $Y = \mu + x_u + x_m + x_g + x_t + x_{my} + \epsilon$, the RMSE of each single effect model is given by:

```
##      predictor model_equation    RMSE
## 1 none (naive)         Y = mu 1.05990
## 2       userId   Y = mu + x_u 0.97840
## 3      movieId   Y = mu + x_i 0.94374
## 4       genres   Y = mu + x_g 1.01780
## 5      genre_1  Y = mu + x_g1 1.04815
## 6      genre_2  Y = mu + x_g2 1.05101
## 7      genre_3  Y = mu + x_g3 1.05626
## 8    timestamp   Y = mu + x_t 1.05620
## 9   movie_year  Y = mu + x_my 1.04910
```

From the table above, the `userId` and `movieId` effects models provide the lowest RMSE. It follows that the `userId` and `movieId` predictors have the greatest impact on the RMSE of the model. The `genres` predictor has a greater overall effect on the RMSE when compared to the separated genres (i.e., `genre_1`, `genre_2`, `genre_3`). The `movie_year` and `genre_1` predictors also have a significant impact on the RMSE of the model. The `timestamp` and `genre_3` predictors have the lowest impact on the RMSE of the model.

If a single-effect model had to be selected, the `movieId` model would be chosen.

Ensemble (multi-effect) Models Since the `timestamp` predictor has the lowest impact on the RMSE of the linear model, it will not be used in the multi-effect models.

```
##                             model_equation    RMSE
## 1                       Y = mu + x_u + x_i 0.88263
## 2              Y = mu + x_g1 + x_g2 + x_g3 1.03832
## 3                 Y = mu + x_i + x_u + x_g 0.88263
## 4                Y = mu + x_i + x_u + x_my 0.88263
## 5 Y = mu + x_i + x_u + x_g1 + x_g2 + x_g3 0.88263
```

From the multi-effect models, it follows that the lowest RMSEs are evident in the models that use a combination of the `userId`, `movieId`, `movie_year` and `genres` predictors. Since there are only 19 unique genres in the `genre_1`, `genre_2` and `genre_3` predictors, the `lm()` function is used in the model $Y = \mu + x_{g1} + x_{g2} + x_{g3}$. The addition of the `genres` predictors has a negligible impact on the RMSE (in comparsion to the `userId` and `movieId` model).

If a multi-effect model had to be selected, any option except the `Y = mu + x_g1 + x_g2 + x_g3` model would be a viable option.

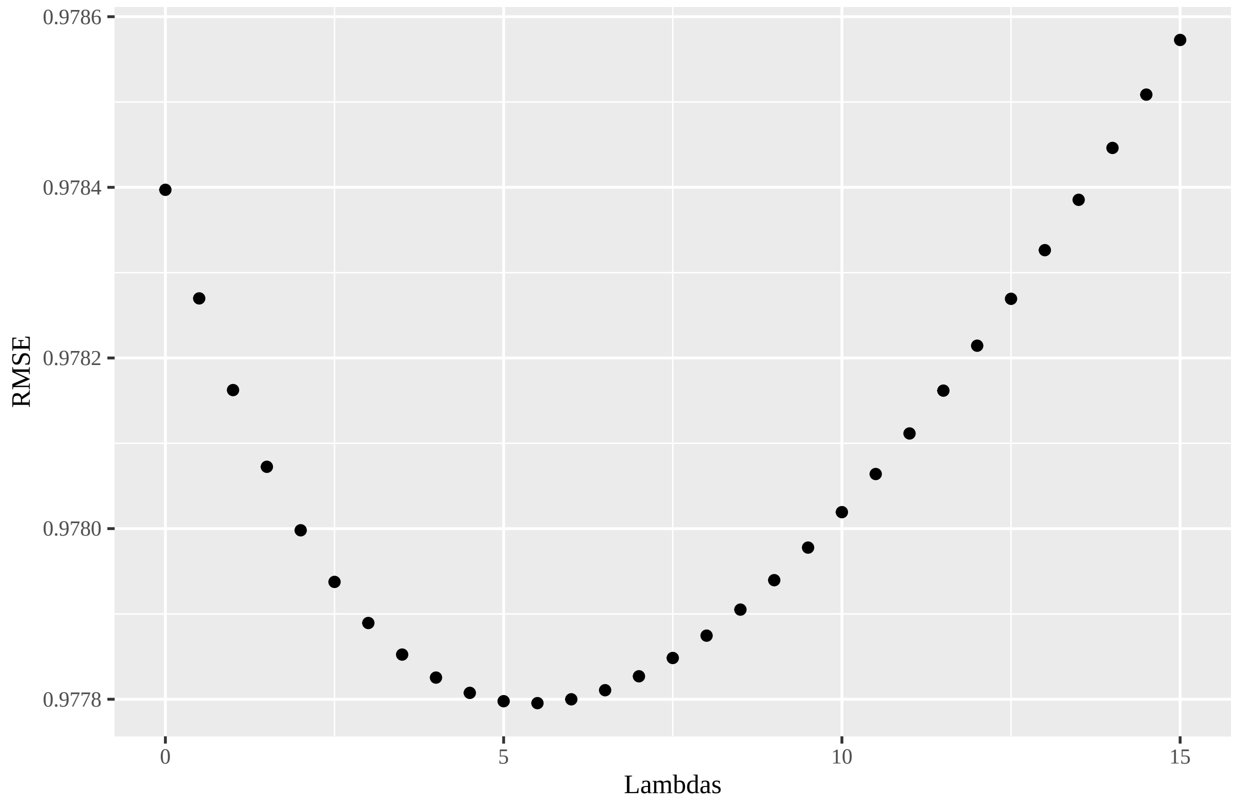
Regularized Models

Regularization is implemented with the tuning parameter λ in increments of 0.5 such that $0 \leq \lambda \leq 10$. It uses the penalized least sum of squares to determine the most effective value of λ for each model:

```r
# Define tuning parameter lambda
lambdas <- seq(0, 10, 0.5)
```

Single Effect Models The results of the following single-effect models are provided in the table below. A sample of the process of choosing the tuning parameter λ is provided in Figure 24 for the `userId` model. From this plot, it follows that the value of λ that provides the lowest RMSE is $\lambda = 5.5$. This method of cross validation is continued for the remainder of the single and multi-effect models.

Figure 24. userId Regularized Model, Lambda Tuning



```
##   model_equation lambda    RMSE
## 1         Y = mu    0.0 1.05990
## 2   Y = mu + x_u    5.5 0.97780
## 3   Y = mu + x_i    2.5 0.94367
## 4  Y = mu + x_my    2.5 1.04910
## 5   Y = mu + x_g    1.0 1.01780
```

From this table, it is evident that regularization has an impact on the RMSE of the model. The RMSE of the `userId` model is `0.97780` (versus `0.97840`, unregularized) and `movieId` model is `0.94367` (versus `0.94374`, unregularized). In the same way as the un-regularized models, the `userId` and `movieId` have the greatest impact on the RMSE of the model. The RMSE of the `movie_year` predictor model has not changed, suggesting that there isn't much noise in the effects of this predictor.

Ensemble (multi-effect) Models

```
##                             model_equation lambda    RMSE
## 1                                   Y = mu    0.0 1.05990
## 2                         Y = mu + x_u + x_i    5.0 0.86524
## 3                   Y = mu + x_u + x_i + x_g    5.0 0.86494
## 4                  Y = mu + x_u + x_i + x_my    4.5 0.86498
## 5 Y = mu + x_u + x_i + x_g1 + x_g2 + x_g3    4.5 0.86542
## 6                  Y = mu + x_u + x_i + x_g1    5.0 0.86514
## 7            Y = mu + x_u + x_i + x_g + x_my    4.5 0.86475
```

From this table, it follows that the best three linear models are:

1. $Y = \mu + x_u + x_i + x_g$
2. $Y = \mu + x_u + x_i + x_{g1}$
3. $Y = \mu + x_u + x_i + x_g + x_{my}$

Summary Comparing the RMSEs of the single-effect and multi-effect models (regularized and unregularized), the multi-effect model $Y = \mu + x_u + x_i + x_g + x_{my}$ will be used for the final model. This model gives the lowest RMSE (`0.86475`), and uses regularization parameter $\lambda = 4.5$.

Selected Model

The selected model is a regularized multi-effect model with the # parameters: $Y = \mu + x_u + x_i + x_g + x_{my}$. It uses the regularization parameter $\lambda = 4.5$, and is implemented with the following code:

```
## Selected Model
# Predictors: User, Movie, Genres, Movie Year
lambda <- 4.5
mu <- mean(edx_train$rating)

movie_effect <- edx_train %>% group_by(movieId) %>%
  summarise(b_i = sum(rating - mu)/(n() + lambda))
user_effect <- edx_train %>% left_join(movie_effect, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - mu - b_i)/(n() + lambda))
movie_year_effect <- edx_train %>% left_join(user_effect, by = "userId") %>%
  left_join(movie_effect, by = "movieId") %>%
  group_by(movie_year) %>%
  summarise(b_my = sum(rating - mu - b_u - b_i)/(n() + lambda))
genres_effect <- edx_train %>% left_join(user_effect, by = "userId") %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(movie_year_effect, by = "movie_year") %>%
  group_by(genres) %>%
  summarise(g_ui = sum(rating - mu - b_u - b_i - b_my)/(n() + lambda))

model <- validation %>% left_join(user_effect, by = "userId") %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(genres_effect, by = "genres") %>%
  left_join(movie_year_effect, by = "movie_year") %>%
  mutate(pred = mu + b_i + b_u + g_ui + b_my) %>%
  pull(pred)

## Validation RMSE
RMSE(validation$rating, model)
```

```
## [1] 0.86516
```

This model gives an RMSE of **0.86475** when tested on the `edx_test` dataset, and **0.86516** when tested on the `validation` dataset.

Conclusion

This report successfully implemented a linear effects machine learning model on the 10M MovieLens dataset. The final linear model used in this report was: $Y = \mu + x_u + x_i + x_g + x_{my}$, and a tuning parameter $\lambda = 4.5$ was used to regularize each predictor. The `userId` and `movieId` predictors consistently had the highest impact on the RMSE of the model. The selected model gives an RMSE of **`0.86475`** when tested on the `edx_test` dataset, and **`0.86516`** when tested on the `validation` dataset. A large portion of the RMSE can be attributed to the fact that the linear model was implemented by approximation: this is one of the main limitations of this project. In the future, the `lm()` function would be used (instead of an approximation) to create a machine learning model. Additional machine learning models could also be used, including dimension reduction (i.e., singular value decomposition) in order to reduce the number of predictors required in the model. The k-nearest neighbors algorithm and regression trees could also be viable options for the improvement of this project In summary, this project was a success.