MovieLens Capstone

C. Hill

11/7/2019

Executive Summary

Results: The RMSE for the validation set is **0.8643**

Model Summary: The model incorporates the following effects on user ratings to make predictions: "Overall Mean Rating" + "Movie Effect" + "User Effect" + "Genre Effect" + "Release Year Effect"

All of these effects are regularized in the final model in order to minimize the impact of a few ratings on the overall prediction.

Introduction

This project uses a data set from the GroupLens research lab (grouplens.org). The entire database includes 20 million ratings from which we take a subset of 10 million ratings. Each row of the data represents a rating given to a movie by one user. We use the subset of 10 million movie ratings to create a training and test set that refine an algorithm to predict what each user would rate a given movie in the database.

Overview

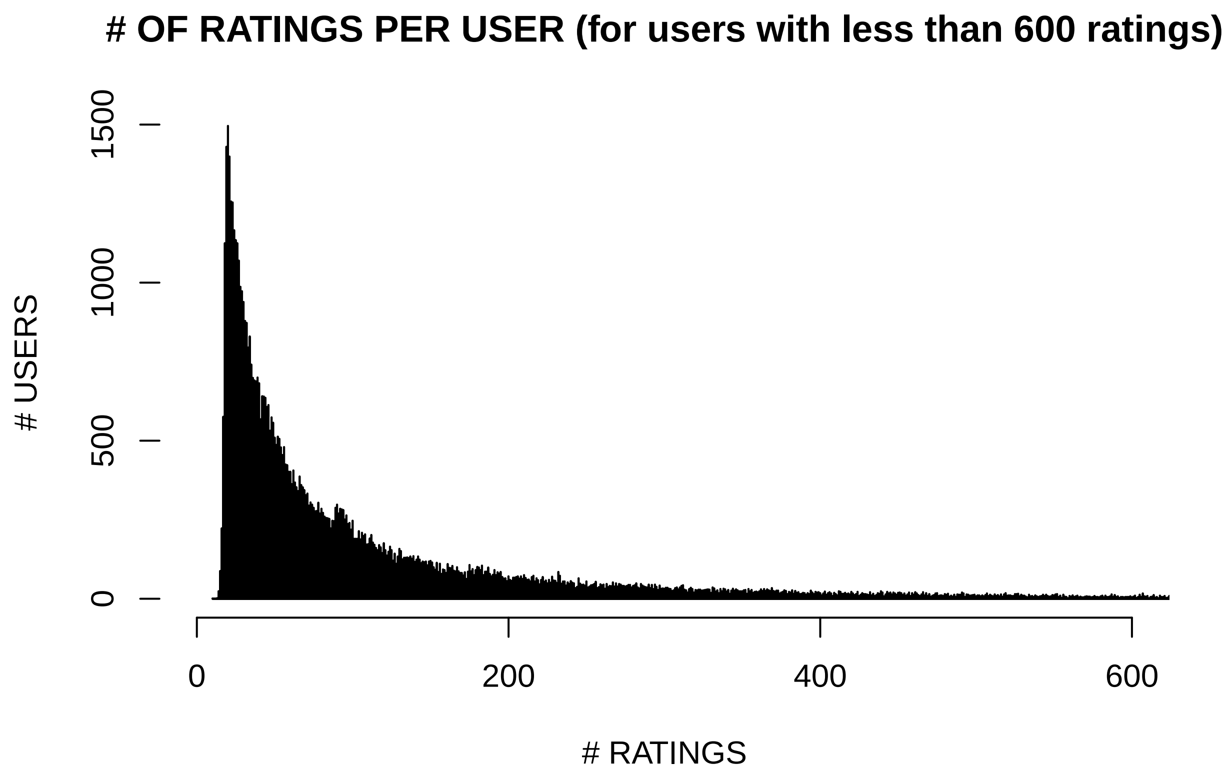
Data Description: The data shows a list of movie ratings (rating) given by users (userId) for various movies. The movies are listed with an ID (movieID), title (title), and genre (genres). A timestamp (timestamp) showing when the rating was given is also included.

For this project, the data is split into a data set (edx) and a validation set (validation). The data set has 9,000,055 ratings (or 90% of the total) while the validation set has 999,999 ratings, (or 10% of the total). Each set has at least one rating from each userID and at least one rating for each movieID. The remaining summary is for the edx data set. This data set will be further split into training and test sets. The validation set is only used to test the final model in the Results section.

Table 1: Edx Summary

	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

The UserID consists of 69,878 users that have contributed the 9 million ratings. The average number of ratings given by a user is 129, while the maximum is a whopping 6,616 ratings and the minimum is a more modest 10 ratings. As can be seen in the figure below, the majority of users give less than 200 ratings with the graph quickly trailing off at higher frequencies.



There are 10,677 movies in the edx dataset with a typical movie receiving 843 ratings. The most popular movie received 31,362 ratings and the least popular 126 movies received 1 rating. The title column includes the release year which ranges from 1915 to 2008. There are quite a few movies with few ratings. Over 78% of movies received less than 600 ratings and 22% of movies received less than 25 ratings. The figures below show the distribution of the number of ratings and the top and bottom 10 films by number of ratings.

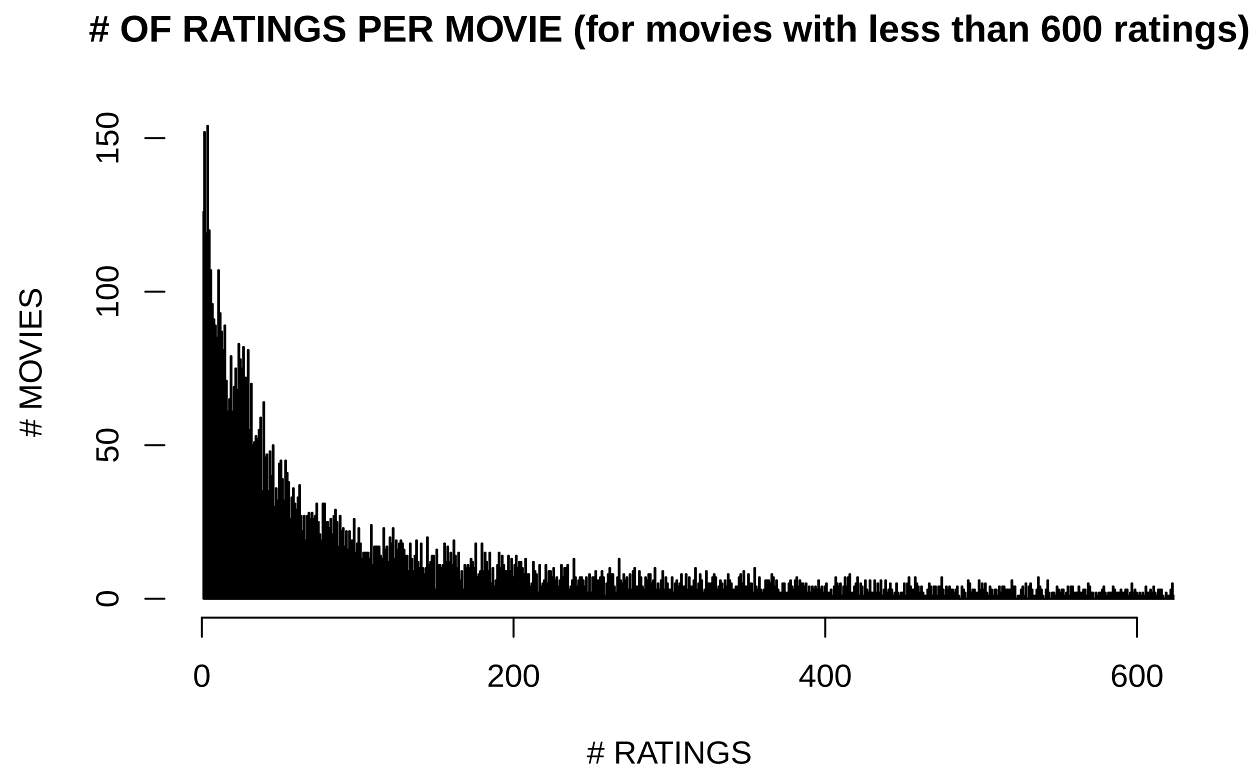


Table 2: Top 10

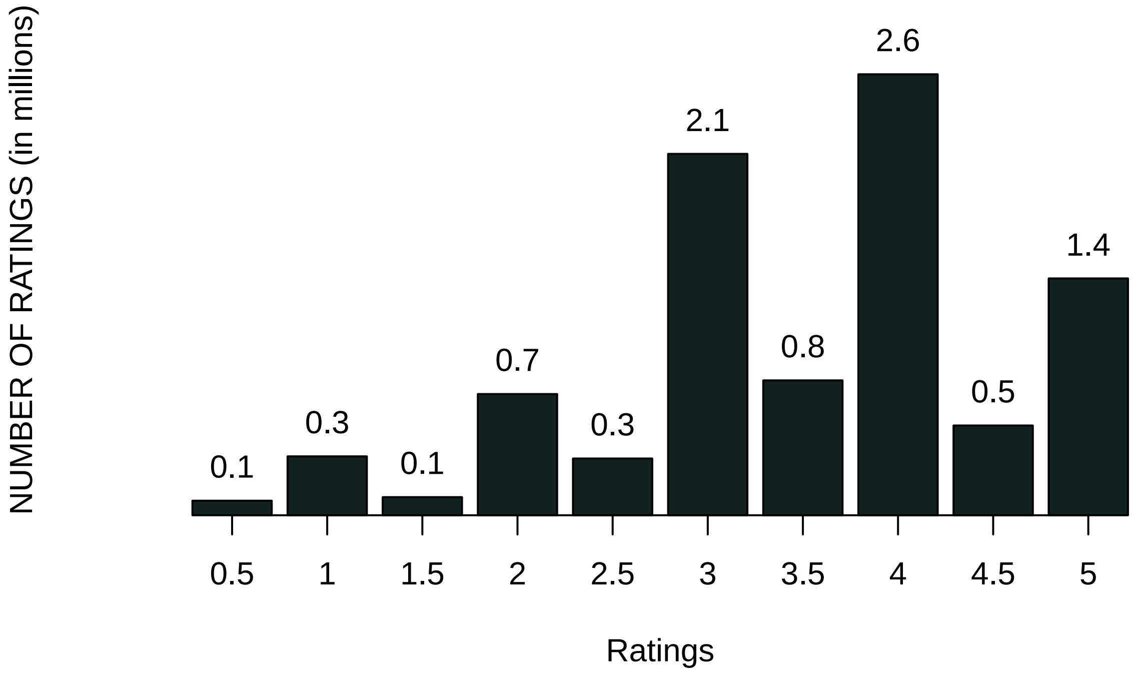
title	n
Pulp Fiction (1994)	31362
Forrest Gump (1994)	31079
Silence of the Lambs, The (1991)	30382
Jurassic Park (1993)	29360
Shawshank Redemption, The (1994)	28015
Braveheart (1995)	26212
Fugitive, The (1993)	25998
Terminator 2: Judgment Day (1991)	25984
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672
Apollo 13 (1995)	24284

The following shows the bottom 10 rows though there are 126 lines that have 1 rating:

Table 3: Bottom 10

title	n
Twice Upon a Time (1983)	1
Uncle Nino (2003)	1
Valerie and Her Week of Wonders (Valerie a tÃ½den divu) (1970)	1
Variety Lights (Luci del varietÃ) (1950)	1
Vinci (2004)	1
When Time Ran Out... (a.k.a. The Day the World Ended) (1980)	1
Where A Good Man Goes (Joi gin a long) (1999)	1
Won't Anybody Listen? (2000)	1
Young Unknowns, The (2000)	1
Zona Zamfirova (2002)	1

The mean rating given for the movie reviews is 3.51 with half ratings (0.5, 1.5, 2.5, 3.5, 4.5) occuring at a lower frequency than whole ratings. The lowest rating available is 0.5 and the highest is 5:

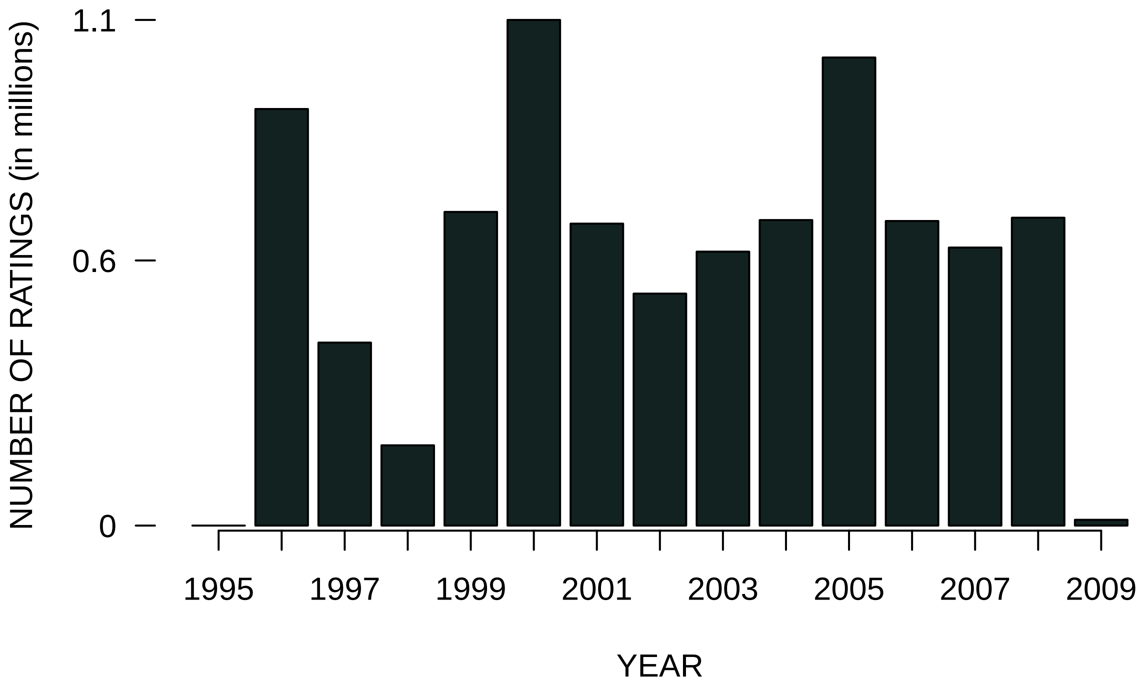


The genres are presented in a list with the potential of more than one genre pertaining to a movie title. For example, The Flintstones (1994) has "Children","Comedy", and "Fantasy" listed for the genre. The table below shows how many movies contain each of the genres.

Table 4: Frequency of Genre

Genres	# of Movies
Drama	3910127
Comedy	3540930
Action	2560545
Thriller	2325899
Adventure	1908892
Romance	1712100
Sci-Fi	1341183
Crime	1327715
Fantasy	925637
Children	737994
Horror	691485
Mystery	568332
War	511147
Animation	467168
Musical	433080
Western	189394
Film-Noir	118541
Documentary	93066
IMAX	8181
(no genres listed)	7

The Timestamp shows the time and date the rating was given in seconds since January 1, 1970. The timespan of the ratings ranges from 1/9/1995 to 1/5/2009.



Project Goal: The provided algorithm creates movie recommendations based on predicted user ratings. The prediction is based on ratings the user has made on other movies with a goal of a minimized RMSE.

Key Steps:

The key steps we take in order to derive the algorithm include the following:

1) *Create a Train and Test set:* The edx data is used to create additional training and test sets to cross validate the model without overtraining to the final validation set.
2) *Clean Data:* The data is reorganized to include the release year in a separate column.
3) *Explore Insights:* Data and anlysis on the correlations between elements of the cleaned data help us pinpoint where to focus our algorithm.
4) *Model Algorithms:* We try different algorithms on the training set.
5) *Evaluate Performance:* We evaluate the performance of the model by predicting ratings for the test set and then comparing our predictions to the actual ratings in the test set.
6) *Finalize Model:* The model producing the lowerst RMSE becomes our final model.
7) *Validate Results:* The final model is tested using the validation set by comparing our predictions to the actual ratings.

Methods/Analysis

Data Cleaning:

1) Create a test set and a training set from the edx data: Our first task is to split the edx data set into a training and test set, with 80% as a training set and 20% as a test set.

```
set.seed(1, sample.kind = "Rounding")

test_index_edx<-createDataPartition(edx$rating,times=1,p = .2, list=FALSE)

edx_test<-edx[test_index_edx,]
edx_train<-edx[-test_index_edx,]

edx_test<- edx_test %>%
semi_join(edx_train, by = "movieId") %>%
semi_join(edx_train, by = "userId")
```

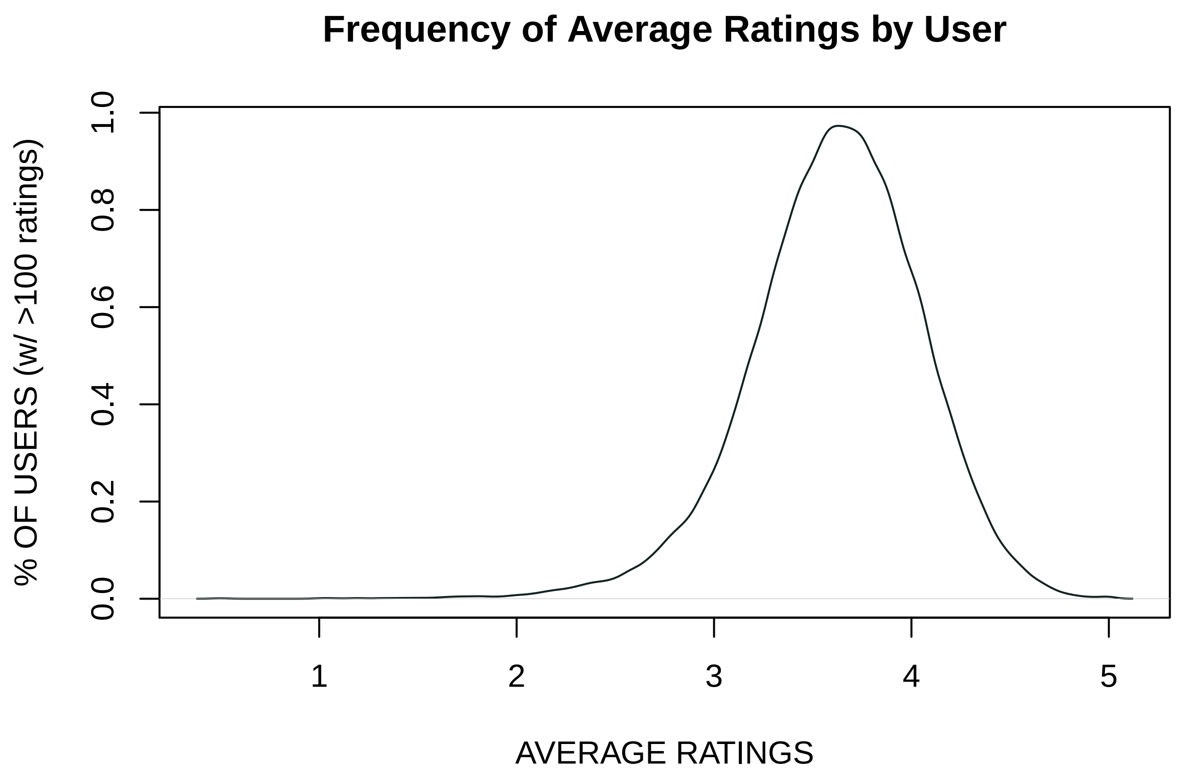
2) Split the Release Year from the Title column into a separate Column

We then extract the release year from the title column into a numeric column named "release_year". The title column will keep the release year within the title in order to differentiate titles that have been remade using the same name. The following code makes this separation:

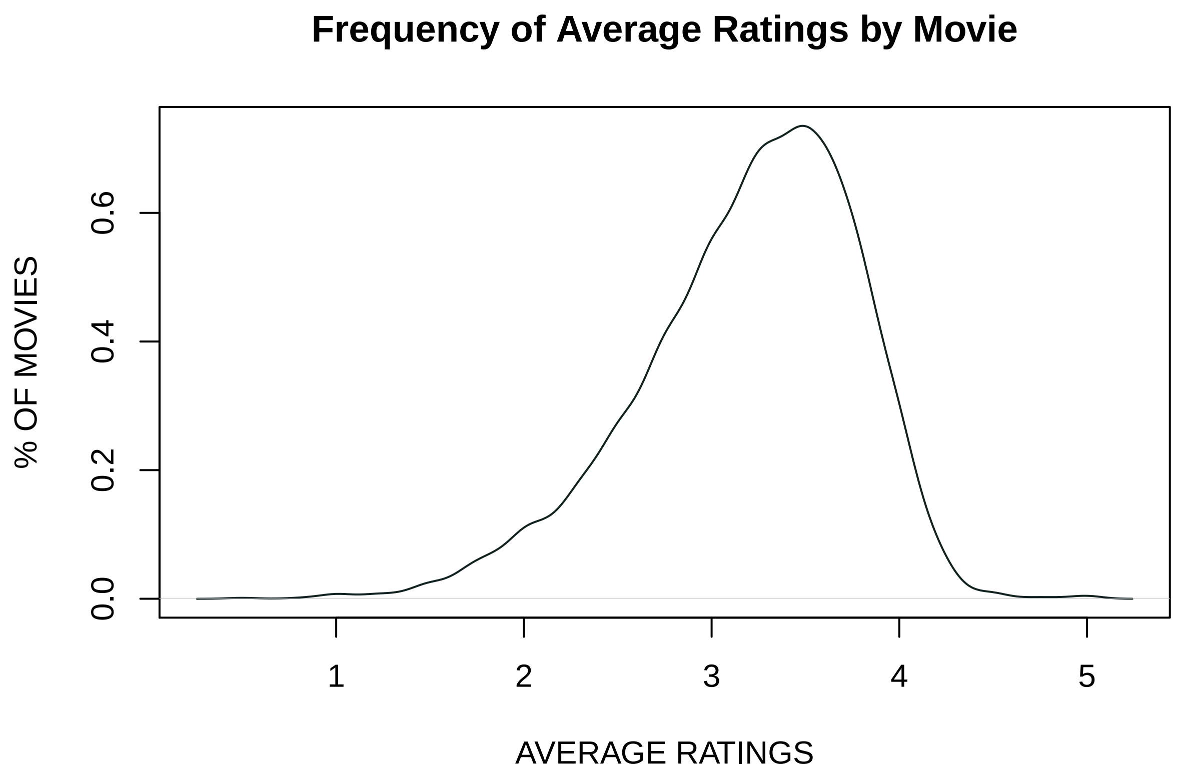
```
years<-str_sub(edx_test$title,start=-6) %>% str_replace_all(.,"\\(|\\)","")
edx_test<-edx_test %>% mutate(release_year=as.numeric(years))

years<-str_sub(edx_train$title,start=-6) %>% str_replace_all(.,"\\(|\\)","")
edx_train<-edx_train %>% mutate(release_year=as.numeric(years))
```

Data Exploration, Visualization, and Insights:

User Effect: Some users are generous with ratings and give out a higher average rating while others are less generous. We see this effect in the following graph showing a density plot of the average rating and the percent of users (with at least 100 ratings) that gave that average rating.



Movie Effect: Different movies will receive either higher or lower average ratings based on the quality of the movie. In order to see this effect, we group the data by title and calculate an average rating for each title. We then create a density plot using only movies with at least 100 ratings showing how common each average rating is. We see that some movies received an average rating of less than 2 and a few movies recieved an average rating over 4.



Genre Effect: In order to show the effect the genre has on movie reviews, we plot the average movie review per genre. In the graph we can see some genres, such as children|Action, generally receive lower ratings than average, while Film-Noir|Mystery generally receives higher ratings. Because of this discrepancy, we will want to include this as part of our model.

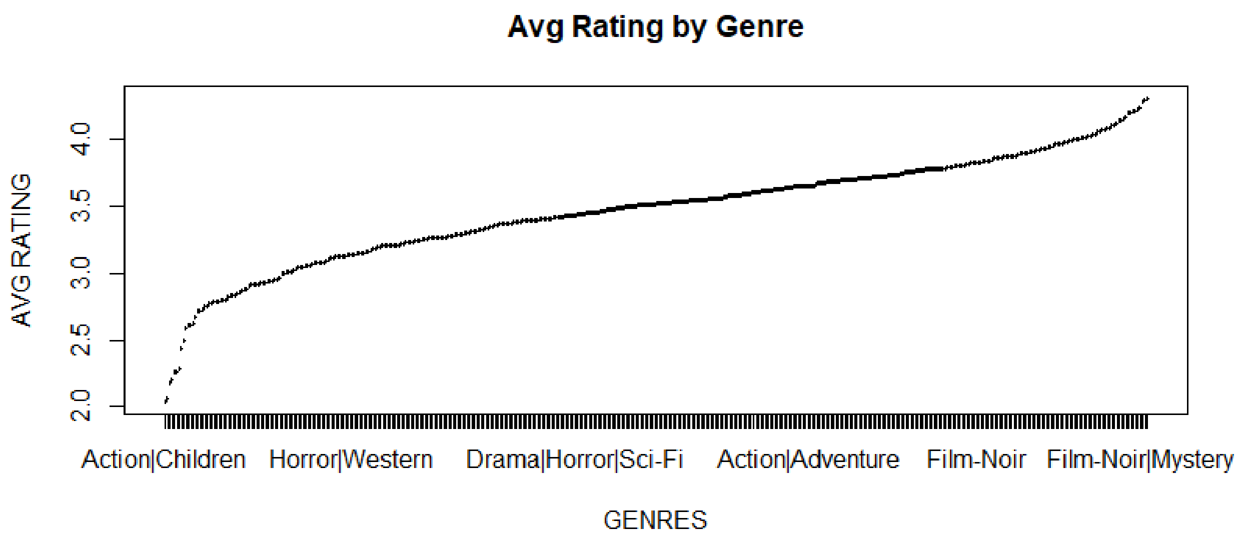
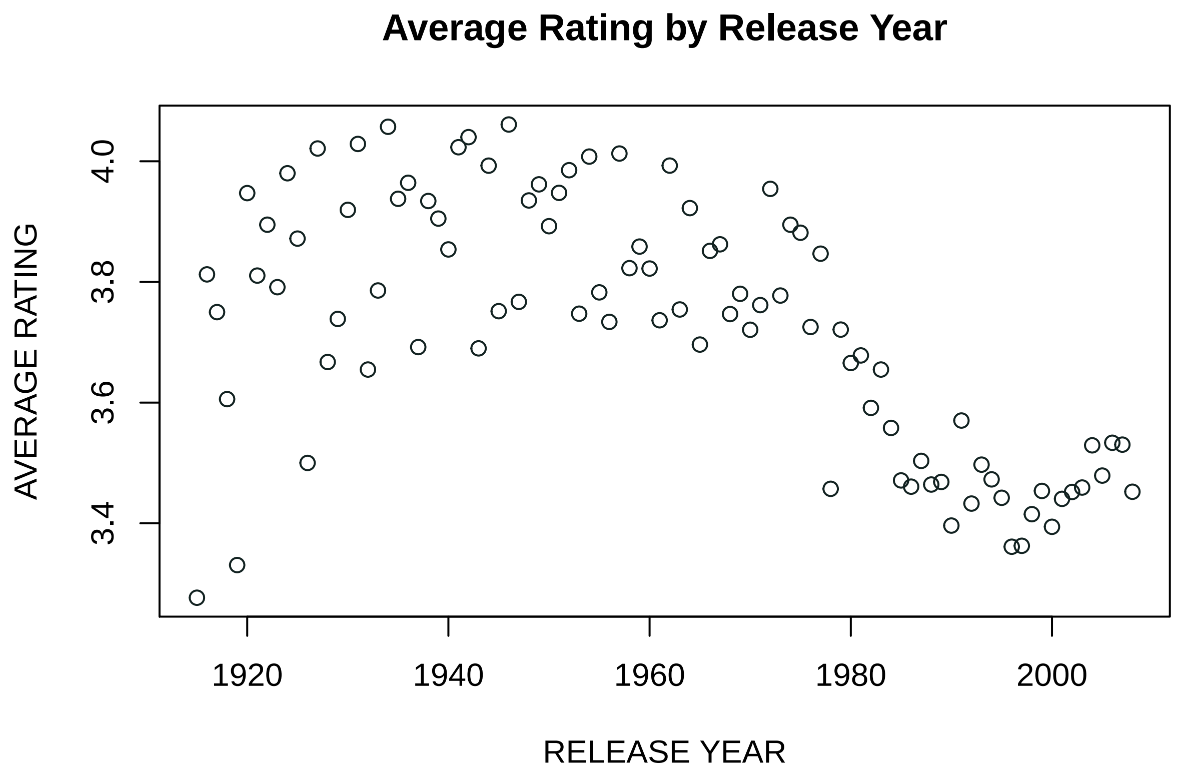
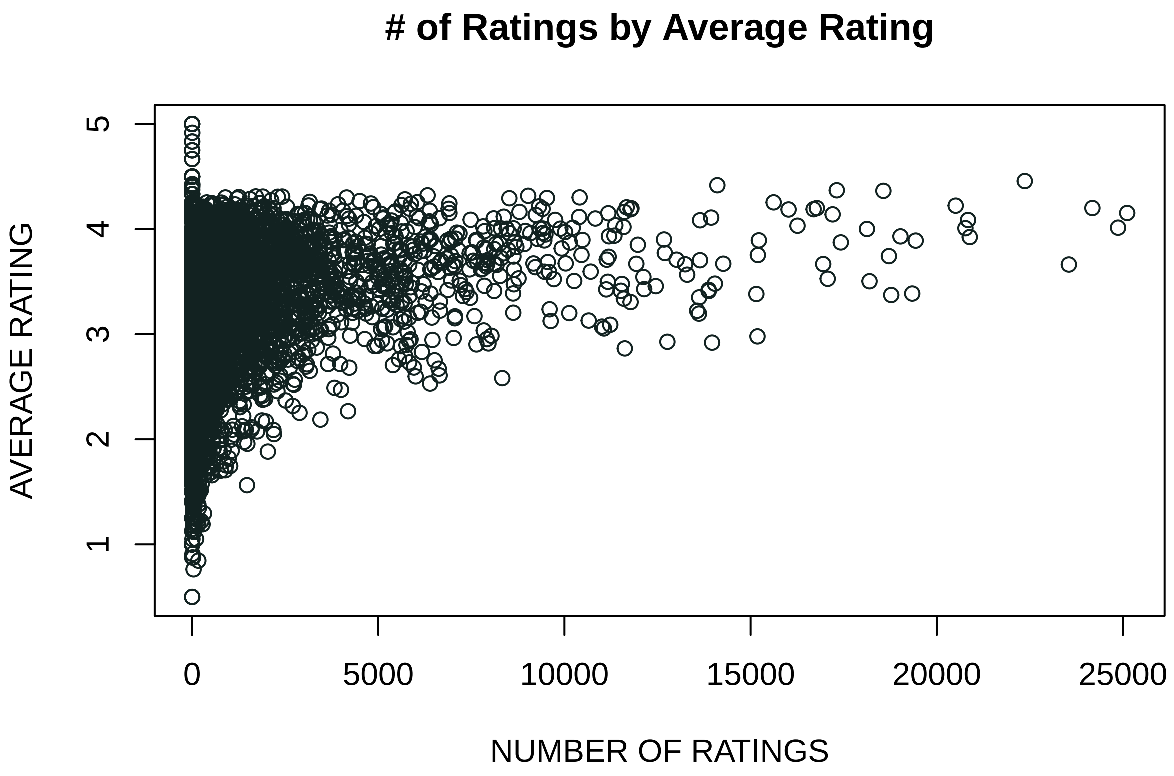


Figure 1: Avg Rating by Genre

Year Effect: The year of release has an effect on movie ratings with newer movies trending more toward the average and older movies having an above average rating.



Number of ratings effect: Many of these effects however, may be overstated if they are based on only a few ratings. For example, we see that the movies with very high or very low average ratings tend to have a lower number of ratings. In order to show this effect graphically, we plot the "Number of Ratings" compared to the "Average Rating". The movies with a higher number of ratings tend to have a mean rating closer to the overall average.



In order to account for this, our model will include a regularization technique on each element of the algorithm described in detail under the Modeling Approach section.

Modeling Approach:

We go through iterations of the model in order to track the error, or RMSE, of the different approaches. In order to test the models, we use the edx_train and edx_test set.

We calculate the model results using the RMSE calculated as follows:

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

Base Model: We start with a base model that predicts the average rating for the unknown rating. The average rating is 3.51 (mu).

```
mu_hat <- mean(edx_train$rating)
Base_RMSE<-RMSE(edx_test$rating,mu_hat)
```

By predicting that each new movie rating will be the same as the average, our RMSE is calculated as: 1.0599.

Movie Effect Model: We calculate the effect of the movie on the average based on the amount the movie's average differs from the overall average. We then predict that any ratings for this movie will differ by the same amount. Following is the code that calculates these differences:

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

predicted_ratings_1 <- mu + edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

Movie_Effect_RMSE<-RMSE(predicted_ratings_1, edx_test$rating)
```

This model offers an RMSE of 0.9437, a strong improvement from using just the averages.

User Effect Model: The user effect model takes into account that different users may tend to give higher or lower average ratings than a typical user. In order to calculate these differences, we use the following code:

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

predicted_ratings_2 <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_u + b_i) %>%
  pull(pred)
Movie_User_Effect_RMSE<-RMSE(predicted_ratings_2, edx_test$rating)
```

The RMSE for this model is 0.8659.

Genre Effect Model: The genre effect model takes into account the fact that different genres tend to receive higher or lower than average ratings. In order to calculate these differences, we use the following code:

```r
genre_avgs <- edx_train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating-mu-b_i-b_u))

predicted_ratings_3 <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_u + b_i + b_g) %>%
  pull(pred)
Movie_User_Genre_Effect_RMSE<-RMSE(predicted_ratings_3, edx_test$rating)
```

The RMSE for this model is 0.8656.

Release Year Model: We also incorporate the effect the release year has on the ratings using the following code:

```
year_avgs <- edx_train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(release_year) %>%
  summarize(b_y = mean(rating-mu-b_i-b_u-b_g))

predicted_ratings_4 <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='release_year') %>%
  mutate(pred = mu + b_u + b_i + b_g + b_y) %>%
  pull(pred)
Movie_User_Genre_Year_Effect_RMSE<-RMSE(predicted_ratings_4, edx_test$rating)
```

This model gives an RMSE of 0.8654. Each model offers a slight improvement. However, we expect that a low number of ratings may skew each of the predictors. For predictors with few ratings, we will predict a rating closer to the mean using the regularization technique in the following section.

Regularized Model: We regularize our model to account for a small number of ratings skewing the results of the model. The regularization predicts a rating closer to the mean if there are only a few ratings to base it on.

We start with an arbitrary lambda of 3 in order to finalize the factors we will keep in our model. In the next step, we optimize lambda to minimize the RMSE. The following code regularizes the model to account for small sample sizes:

```
lambda <- 3

mu <- mean(edx_train$rating)

 movie_avgs <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating-mu)/(n() + lambda))

 user_avgs <- edx_train %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating-mu-b_i)/(n() + lambda))

 genre_avgs <- edx_train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating-mu-b_i-b_u)/(n() + lambda))

 year_avgs <- edx_train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(release_year) %>%
  summarize(b_y = sum(rating-mu-b_i-b_u-b_g)/(n() + lambda))

predicted_ratings_5 <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='release_year') %>%
  mutate(pred = mu + b_u + b_i + b_g + b_y) %>%
  pull(pred)
Movie_User_Genre_Year_Reg_Effect_RMSE<-RMSE(predicted_ratings_5, edx_test$rating)
```

The resulting RMSE is 0.8649.

We choose a lambda by iterating through different lambdas to minimize the RMSE. We create a training and test set from our original edx dataset in order not to train on our final validation set. The following code creates these sets:

```
set.seed(1, sample.kind = "Rounding")

edx_sample<-sample_n(edx,100000)
lambda_test_index<-createDataPartition(edx_sample$rating,times=1,p = .2, list=FALSE)

lambda_test<-edx_sample[lambda_test_index,]
lambda_train<-edx_sample[-lambda_test_index,]

lambda_test<- lambda_test %>%
  semi_join(lambda_train, by = "movieId") %>%
  semi_join(lambda_train, by = "userId")

years<-str_sub(lambda_test$title,start=-6) %>% str_replace_all(.,"\\(|\\)","")
  lambda_test <- lambda_test %>% mutate(release_year=as.numeric(years))

years<-str_sub(lambda_train$title,start=-6) %>% str_replace_all(.,"\\(|\\)","")
  lambda_train <- lambda_train %>% mutate(release_year=as.numeric(years))
```

We use this test set to iterate through different lambdas to see which lambda optimizes our RMSE. The following code allows us to pinpoint lambda:

```
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu <- mean(lambda_train$rating)

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

   user_avgs <- lambda_train %>%
    left_join(movie_avgs, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating-mu-b_i)/(n() + l))
```

```r
   genre_avgs <- lambda_train %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating-mu-b_i-b_u)/(n() + l))

   year_avgs <- lambda_train %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(genre_avgs, by='genres') %>%
    group_by(release_year) %>%
    summarize(b_y = sum(rating-mu-b_i-b_u-b_g)/(n() + l))

  predicted_ratings_7 <- lambda_test %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(genre_avgs, by='genres') %>%
    left_join(year_avgs, by='release_year') %>%
    mutate(pred = mu + b_u + b_i + b_g + b_y) %>%
    pull(pred)

  return(RMSE(predicted_ratings_7, lambda_test$rating))
})

plot(lambdas, rmses)
lambdas[which.min(rmses)]
```

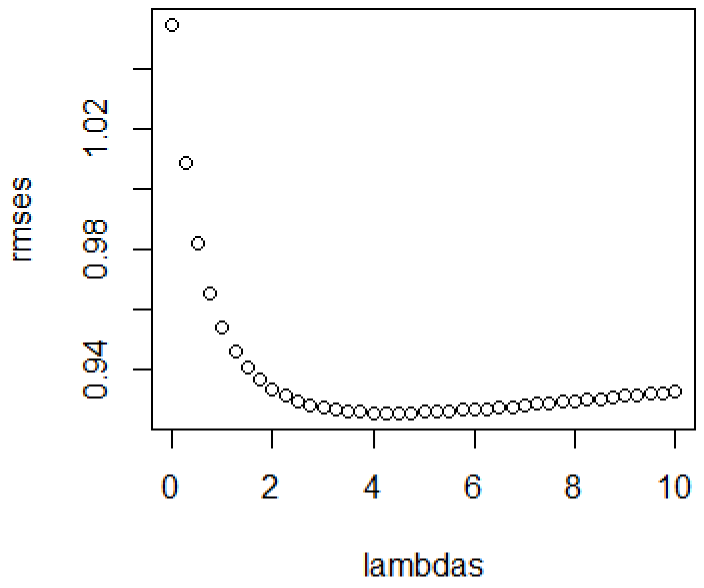


Figure 2: RMSE per Tested Lambda

We see the lambda that optimizes the RMSES on the lambda test set is 4.25. We use this lambda to calculate the RMSE for our models.

Results

Model Results:

The following table summarizes the RMSE acheived using the different modeling methods and a lambda of 4.25:

Table 5: RMSE Results Summary

method	RMSE
Base Model	1.0599043
Movie Effect	0.9437429
Movie + User Effect	0.8659320
Movie + User + Genre Effect	0.8655941
Movie + User + Genre + Year Effect	0.8654189
Movie + User + Genre + Year Effect Regularized	0.8647994

Model Performance:

In order to determine the final performance of the model, we use the original edx and validation sets. The following code uses this data to calculate the RMSE of the final model:

```
#The first step is to create a "Release Year"
#column in the validation and edx sets for the code to function

  years<-str_sub(validation$title,start=-6) %>% str_replace_all(.,"\\(|\\)","")
  validation <- validation %>% mutate(release_year=as.numeric(years))

  years<-str_sub(edx$title,start=-6) %>% str_replace_all(.,"\\(|\\)","")
  edx <- edx %>% mutate(release_year=as.numeric(years))

#We then use our optimized lambda from the test sets and run
#the code using the edx and validation sets.

      lambda <- 4.25

      mu <- mean(edx$rating)

      movie_reg_avgs <- edx %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating-mu)/(n() + lambda))

      user_reg_avgs <- edx %>%
        left_join(movie_reg_avgs, by="movieId") %>%
        group_by(userId) %>%
        summarize(b_u = sum(rating-b_i-mu)/(n() + lambda))

      genre_reg_avgs <- edx %>%
        left_join(movie_reg_avgs, by="movieId") %>%
        left_join(user_reg_avgs, by = "userId") %>%
        group_by(genres) %>%
        summarize(b_g = sum(rating-b_i-b_u-mu)/(n() + lambda))

      year_reg_avgs <- edx %>%
        left_join(movie_reg_avgs, by= "movieId") %>%
        left_join(user_reg_avgs, by = "userId") %>%
        left_join(genre_reg_avgs, by = "genres") %>%
        group_by(release_year) %>%
        summarize(b_y = sum(rating-b_i-b_u-b_g-mu)/(n() + lambda))

      predicted_ratings <- validation %>%
        left_join(movie_reg_avgs, by= "movieId") %>%
        left_join(user_reg_avgs, by = "userId") %>%
        left_join(genre_reg_avgs, by = "genres") %>%
        left_join(year_reg_avgs, by = "release_year") %>%
        mutate(pred = mu + b_i + b_u + b_g + b_y) %>%
        pull(pred)

      round(RMSE(predicted_ratings, validation$rating),4)
```

```
## [1] 0.8643
```

Conclusion

Summary:

By incorporating the “Overall Mean Rating” + “Movie Effect” + “User Effect” + “Genre Effect” + “Release Year Effect” we achieve an overall RMSE for the validation set of **0.8643**. This signifies that the algorithm will error on average by .8643 stars.

Limitations of this algorithm are its ability to model how groups of users and groups of movies relate to each other. For example, users that like movies with Sandra Bullock may like other movies with her as an actress. Due to the number of different factors in the data that may have relationships to eachother, the algorithm can be honed further in the future by incorporating factor analysis to account for these relationships.