PH125.9x Data Science Capstone

MovieLens Recommender System

Adam C. Riley

11/10/2020

The R Markdown code used to generate this pdf document and associated files can be found my Github.

Capstone Executive Summary

Overview

Recommender systems are a branch of artificial intelligence designed to predict the rating a specific user would give an unknown or newly suggested item by using machine learning algorithms. These types of systems are very popular and are the driving force behind recommendations provided to users of platforms like Amazon, YouTube and Netflix.

This Capstone Project involves the creation of a recommender system for movies and is loosely is designed to mimic a famous competition ("The Netflix Prize Challenge") in which teams competed to predict user ratings on films without direct information about the movie or user. Identifying information about users and movies was masked and assigned random numbers. Participating teams were only given the rating the user provided to each movie and some general classifier data on the film. While simple recommender systems are prevalent, making improvements to a robust existing system can be very difficult after a certain point; Netflix awarded the winning team a million dollars for a roughly 10% increase in accuracy for their algorithm.

Data

We do not have access to the proprietary Netflix data, but we can use similar data collected by GroupLens Research, a research lab in the Department of Computer Science and Engineering at the University of Minnesota. GroupLens Research has collected film preference information on users into a data set called MovieLens, containing millions of ratings on tens of thousands of different movies. There are also hundreds of thousands of unique user profiles for the contributors. Users assign a rating of 1 - 5 for each movie, with 5 being the highest rating for movies they enjoyed and 1 being low rating for a movie they did not like. Users can assign values at either whole number or half number values (e.g. 1.5, 2, 3.5, 9or 4). Our goal is to try and predict the rating a user would give a new movie presented to them. we will be building the recommender utilizing a preformatted version the 10M set of the MovieLens data.

Process

I will follow the following steps to create the model for this project:

1. Import Course Data
2. Initial Exploration of Data
3. Prepare the Data
4. Deep Dive
5. Create Models
6. Evaluation
7. Results

Models

This analysis will focus on linear models and regularized linear models.

Note: There were points in the analysis where I tried other models, but did not have the memory or computing power to create them. I have left the code available for you to run if you would like, but I will not go into detail about the methodology or results.

Linear Models

The models are simple but are widely used and can be effective. Linear model presume a straight line relationship between the factors and make predictions based on that assumption.

Linear models fit the general form of:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + ... + \epsilon_{i,u}$$

Where $\hat{Y}$ is the predicted rating, μ is the true mean of ratings, b_i and b_u is a collection of terms are the true bias effects of the features selected, and $\epsilon_{i,u}$ is the error distribution. We'll begin our investigation with no selected features and build out more complex models as we explore.

With no selected features, the predicted value is simply the mean:

$$\hat{Y}_{u,i} = \mu + \epsilon_{i,u}$$

We will also look at adding the the individual effect of each factor we select. Those individual bias effects b_i take for the form of:

$$\hat{b}_i = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{\mu})$$

where $\hat{b}_i$ is the predicted value of b_i, N is the number of ratings, y_i is the rating, and $\hat{\mu}$ is the predicted mean. Since we aim to measure a particular feature, we treat that each item in that category as a small set. We remove take out our initial model and then calculate the mean across the deviation from the original prediction. The mean of the deviations is used to correct our prediction for all items in the category.

If we can combine two or more features together to estimate the bias of further terms:

$$\hat{b}_u = \frac{1}{N}\sum_{i=1}^{N}(y_{u,i} - \hat{b}_i - \hat{\mu})$$

where $\hat{b}_u$ is the prediction for the second factor, N is the number of ratings, $y_{u,i}$ is the rating, $\hat{b}_i$ is the predicted bias from the first feature, and $\hat{\mu}$ is the predicted mean. We do the exact same math here, picking a second category but this time removing the original prediction and the modified second prediction. After those predictions are removed, we are again left with a set of deviations based on a feature and then we can assign the mean of those deviations for the feature. We can continue to add terms in this way and we will increase our accuracy each time.

One thing to note here, while we can continue to increase our accuracy for the whole set, at some point the features we select will have such small groups that they might contain only 1 member. This could lead to a model that is so precise it will not be able to account for new variations. This is called 'over-fitting.'

Regularized Linear Models

Regularized linear models are similar to linear models and the basic structure and stepwise building approach outline in the previous section is a core part of this model. When we look at purely linear models however, they can be skewed by the number of ratings for individual entries within a category with these entries contributing hugely to the estimate, causing predictors that have very few entries to have larger estimated errors. To address this, we can 'regulate' the data by adding a small term that penalizes small sample sizes and discounts them accordingly but won't be impactful for larger and more accurate group terms. We will call this small term λ. This is how we'd add it to the multi-factor linear model referenced above:

$$\hat{b}_u = \frac{1}{n_u + \lambda} \sum_{i=1}^{n_u} (y_{u,i} - \hat{b}_i - \hat{\mu})$$

Where again, $\hat{b}_u$ is the prediction for the second factor, n_u is the number of ratings for the factor, $y_{u,i}$ is the rating, $\hat{b}_i$ is the predicted bias from the first feature, and $\hat{\mu}$ is the predicted mean.

You see that for large groups of n_u the λ term will not cause a large variation in the result because the ration will be similar. For small values of n_u, this would cause the term to be less important. There are several values of λ we could choose and each approximation will discount some groups more than other. We to need to will test several values of λ and select a value that minimizes our model error.

Metrics

The accuracy of the recommender will be evaluated using a Root Mean Square Error calculation (RMSE) which measures the difference between the prediction and actual value assigned by the user. RMSE is defined as:

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

where N is the number of ratings, $y_{u,i}$ is the rating of movie i by user u and $\hat{y}_{u,i}$ is the prediction of movie i by user u. This metric is designed to minimize the penalty for small deviations from the true rating but to amplify large errors. This is good measure for a recommender system with discrete user predictions because small errors are acceptable and do not change our final recommended rating on the whole or half number close to our prediction while still focusing on minimizing large deviations.

We will use the provided data to create a model that has an accuracy with a RMSE < 0.86490.

Import Course Data

R Packages

Before we get started, we will need to load several libraries into R. All packages can be downloaded from CRAN.

- Packages required for data import
  - tidyverse
  - caret
  - data.table
- Packages required for data analysis
  - lubridate
  - dplyr
  - reshaper2
  - ggplot2

Loading Preformatted Data

Now that we have tools installed, we will ingest the data from GroupLens and perform the preprocessing required by the course. This formats the data consistently for all students.

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

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

Once imported the data is split into two sub-sets from the main set. The first, approximately 90% of the set designated as **edx**, is for the student's machine learning processing and the second, referred to as **validation**, is to be used by student's as a final test of our models. The items in the **validation** set are verified against the **edx** set so that there are no users or movies that are unknown in the testing set.

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

After the data is imported, we remove unnecessary temporary collections to clean up our workspace.

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

Intial Exploration of Data

We will start by just looking at the format of the data to see what is provided and what we can extract to use for modeling:

```
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 42...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)",...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|...
```

We can see that there 6 columns and a lot of potential that needs to be extracted before we can take a deeper dive.

Firstly, we can see that users and movies have been given unique identifiers that should be easy to manipulate but the time stamp is not in a standard format and need to be converted to help make sense of it during our analysis steps.

Secondly, we can see that movie titles contain strings of text with the year the movie was released set off in parentheses. We'll need to capture that portion of the movie title and then strip off the parentheses. This two-step method will be used instead of just attempting to isolate the 4 digit year directly because some movie titles include 4 digit numbers in the title itself and I do not want to get the wrong number.

Next, because we are only able to get the year the movie was released, I also want to highlight the year the movie was reviewed apart from the more specific date. We can use this to determine the difference between when the movie was released and the year it was reviewed. My hypothesis is that this will be valuable because movies that are more popular should continued to be reviewed as time goes on and that these reviews will be more positive. We will explore this later.

Finally, we can see that the genre information includes strings of one and sometimes more than one genre. I have another hypothesis that individual genres might be more meaningful than combination, so I also want to split the information in that column apart so that we can explore that during our deep dive also.

Prepare the Data

Now that we know what the data needs to look like, we need to manipulate it to match.

Dates

The first few steps involve converting the dates into the usable formats we mentioned above, adding columns for 'reviewyear' and 'reviewday' which we will populate using simple conversions from the existing 'timestamp' field:

```
reviewyear = year(as_datetime(timestamp))
reviewday = format(as_datetime(timestamp), "%Y-%m-%d")
```

Then we will use the string extraction function to grab the portion of the movie title that is formatted like '(###)' and will then strip off the parenthesis using a two-step method:

```
releaseyear = str_extract(str_extract(title, regex("\\((\\d{4})\\)")), regex("(\\d{4})"))
```

Finally, we will take the date information we have collected so far and calculate the difference between the release year and the reviewed year to get an estimate of how old the movie was when it was reviewed:

```
reviewdelta = (as.numeric(reviewyear) - as.numeric(releaseyear))
```

This additional information can be added to the both the **edx** data in one command, and then the **validation** set using identical commands.

```
edx <- edx %>% mutate(
ID = seq.int(nrow(edx)),
reviewyear = year(as_datetime(timestamp)),
reviewday = format(as_datetime(timestamp), "%Y-%m-%d"),
releaseyear = str_extract(str_extract(title, regex("\\((\\d{4})\\)")), regex("(\\d{4})")),
reviewdelta = (as.numeric(reviewyear) - as.numeric(releaseyear))
)
```

Genres

We need to a little of pre-work before we can parse the genres in a machine learning friendly way. Because we know that all the movies in the **validation** set are included in the **edx** set, we will just focus on manipulating the **edx** movies first and they we can apply the results to both sets.

We begin by separating each individual genre onto its own line based on the pipe separator:

```
movie_genres_temp <- edx %>% separate_rows(genres, sep = "[|]")
```

Now we have a single column which contains a list all the genres from all the movies. This has several duplicate entries and we will remove them so that we have a list that contains each genre listed in the **edx** set:

```
movie_genres <- unique(movie_genres_temp$genres)
movie_genres
```

```
##  [1] "Comedy"             "Romance"            "Action"
##  [4] "Crime"              "Thriller"           "Drama"
##  [7] "Sci-Fi"             "Adventure"          "Children"
## [10] "Fantasy"            "War"                "Animation"
## [13] "Musical"            "Western"            "Mystery"
## [16] "Film-Noir"          "Horror"             "Documentary"
## [19] "IMAX"               "(no genres listed)"
```

With the list of genres, we will loop through and a column to our data for each genre that has been identified. We will then check to see if that genre is in the genre list of each movie. If we find the genre, we will mark that entry with a 1 in that column and will mark it with a 0 if it was not.

```
for(i in seq_along(counting)) {
    edx <- add_column(edx, !!movie_genres[i] := as.numeric(str_detect(edx$genres, movie_genres[i])))
    validation <- add_column(validation, !!movie_genres[i] := as.numeric(str_detect(validation$genres, m
}
```

We have now extracted all the additional information we need to take our deep dive.

Split the Data into Training and Testing Sets

We will follow the same methodology used above, and split the **edx** set into training and testing sub sets.

```
set.seed(61, sample.kind = 'Rounding')
train_index <- createDataPartition(edx$ID, times = 1, p = 0.2, list = FALSE)
edx_train_set <- edx[-train_index,]
temp_test_set <- edx[train_index,]
```

We again validate the data to ensure that there are no surprises.

```
edx_test_set <- temp_test_set %>%
      semi_join(edx_train_set, by = "movieId") %>%
      semi_join(edx_train_set, by = "userId")
removed <- anti_join(temp_test_set, edx_test_set)
edx_train_set <- rbind(edx_train_set, removed)
```

And we should clean up the data to select just the columns we intend to use and make sure the formats are usable.

```
tidy_validation_set <- select(validation, userId, movieId, releaseyear, rating, reviewday, reviewyear, r
tidy_test_set <- select(edx_test_set, userId, movieId, releaseyear, rating, reviewday, reviewyear, revie
tidy_train_set <- select(edx_train_set, userId, movieId, releaseyear, rating, reviewday, reviewyear, rev
```

```
tidy_validation_set <- tidy_validation_set %>% mutate(userId=as.integer(userId), movieId=as.integer(movi
tidy_test_set <- tidy_test_set %>% mutate(userId=as.integer(userId), movieId=as.integer(movieId), releas
tidy_train_set <- tidy_train_set %>% mutate(userId=as.integer(userId), movieId=as.integer(movieId), rel
```

Finally, we will again remove unnecessary temporary collections to clean up our workspace again.

```
rm(movie_genres_temp, counting, i, edx, train_index, temp_test_set, removed, validation, edx_test_set, 
```

Deep Dive

From previous course work, I know that both user preference and movie selection are meaningful and will be included in the models. Because of that, this analysis will focus on exploring the effect of dates and the effects of genres.

GENRES

Single Genre Effects

My first thoughts after setting up the data was to jump in and determine the importance of each genre using a glm model. This would let us model all of the genres as linear factors and we could see the overall importance of each by looking at the coefficient generated. I started by melting the data using the reshaper2 package which tidied the data into 3 columns with rating, genre, and a 1 or 0 indication if the genre is present in the movie's genre list.

```
## Rows: 144,001,620
## Columns: 3
## $ rating   <dbl> 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0...
## $ variable <fct> Comedy, Comedy, Comedy, Comedy, Comedy, Comedy, Comedy, Co...
## $ value    <dbl> 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1...
```

This is an excellent formatting for machine learning, but the data get get a lot longer. When I attempted to run the resulting collection through a simple glm model fit, it failed.

```
x <- select(tidy_train_set, -rating)
set.seed(61, sample.kind = 'Rounding')
glmfit <- train(x, tidy_train_set$rating, method="glm")
1: model fit failed for Resample01: parameter=none Error : cannot allocate vector of size 36.4 Gb
```

After that, I decided to use a visual inspection, but was also unable to plot the data due to size restrictions.

```
bigplot <- ggplot(data = melted_genres, aes(x=variable, y=rating)) + geom_boxplot(aes(fill=as.factor(va
bigplot + facet_wrap( ~ variable, scales="free")
Error: cannot allocate vector of size 549.3 Mb
```

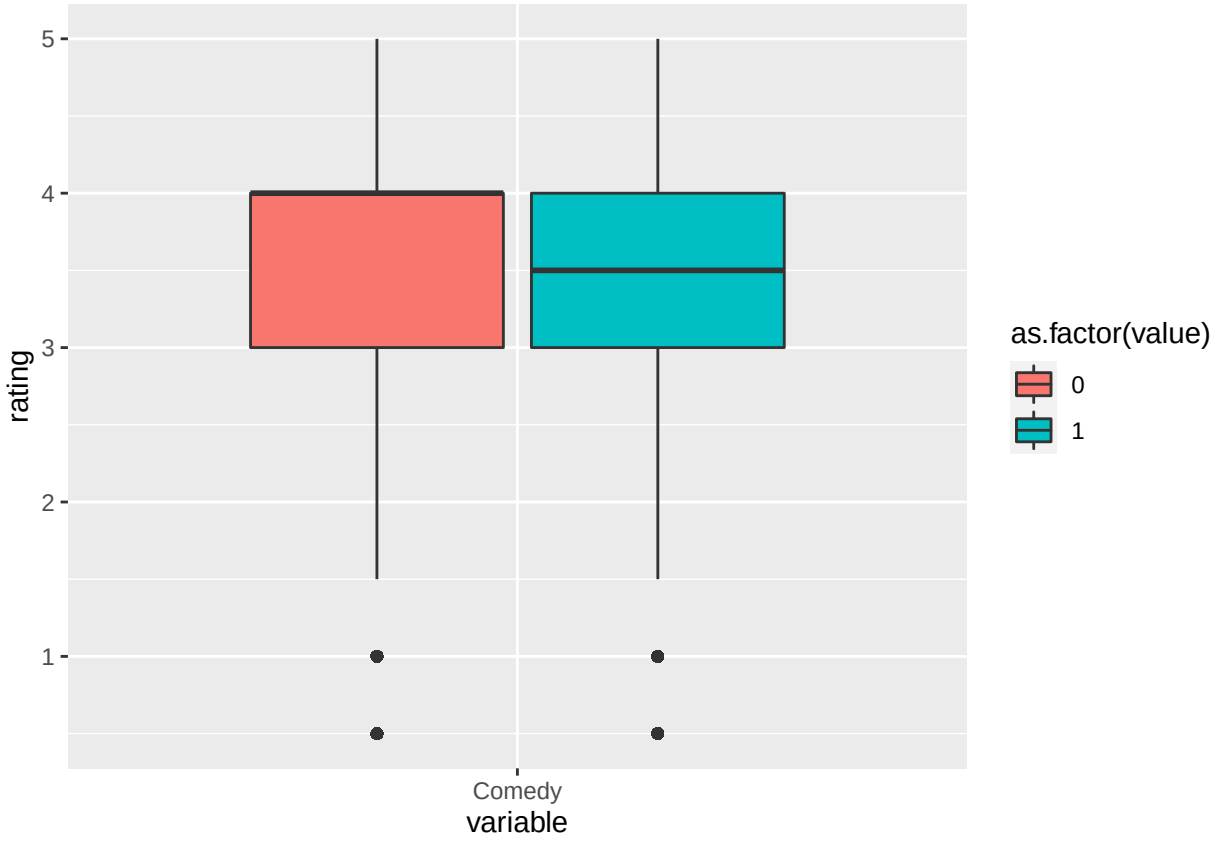
So even though I could not model them all together, I wanted to try them one at a time. I was still determined to investigate the data in this way and decided to by looking at a single genre, I could figure out if the approach would work and then extrapolate if I the results seemed promising.

We can pick out a genre to start with by seeing which genres are the most popular. First we will just look at a distinct list of all of the movies we are examining and then we'll total the number of hits in each of the genre columns:

```
##              Drama             Comedy           Thriller            Romance
##               5336               3703               1705               1685
##             Action              Crime          Adventure             Horror
##               1473               1117               1025               1013
##             Sci-Fi            Fantasy           Children                War
```

```
##             754             543             528             510
##         Mystery     Documentary         Musical       Animation
##             509             481             436             286
##         Western       Film-Noir            IMAX (no genres listed)
##             275             148              29               1
```

I selected the comedy genre because it was near but not at the top, and again melted the data. This time with a filter to focus the set for comedy movies in hopes of getting a manageable data set. Before we move on, we will compare the distribution of films that have the comedy genre tag vs those that do not.



While there is quite a difference visually between the means, these charts are almost identical. I believe this is because movies with higher ratings are more popular and would be ranked much more often than movies with lower ratings. This would heavily skew the results.

I needed a simpler way to example these effects.

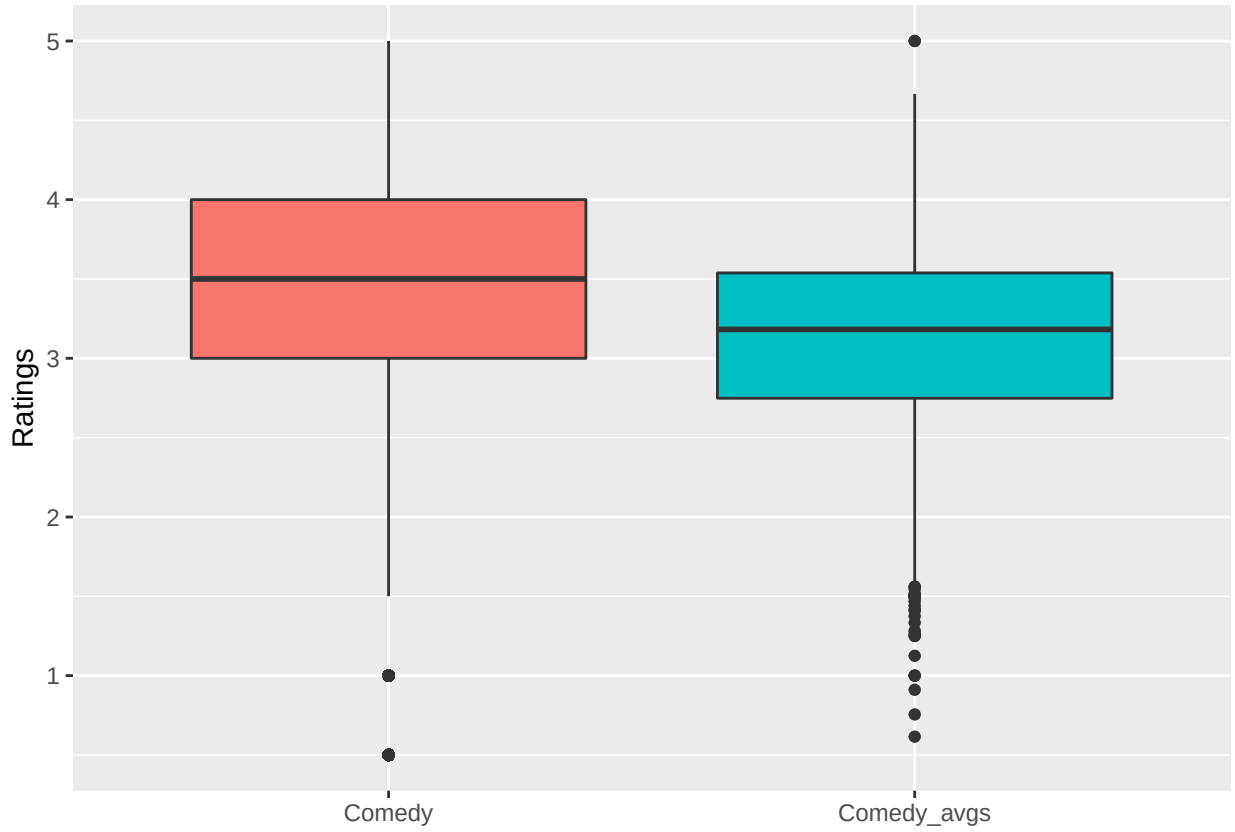
I decided it would be best to view the a compressed version of the data and will focus on the average the rating for each film.

```
movie_avgs <- tidy_train_set %>% group_by(movieId) %>% summarize(b_mu = mean(rating))
movie_avgs <- arrange(movie_avgs, movieId)
genres_train_set <- arrange(genres_train_set, movieId)
genres_train_set$avgrating <- movie_avgs$b_mu
```

Before we get too far down this road, we need to step back and see how this has choice impacts out distribution of data. Since we already have the comedy data separated, let's take a look at how pulling out the averages changes the spread of the data. My assumption is that it should lower the mean and decrease the quartile range.

```
comedy_avgs <- genres_train_set %>% filter(Comedy == 1) %>% select(avgrating)
comedy_avgs$rating <- comedy_avgs$avgrating
comedy_avgs$variable <- "Comedy_avgs"
comedy_raw <- melted_comedy %>% filter(value == 1) %>% select(rating, variable)
comedy_combo <- rbind(comedy_raw, select(comedy_avgs, rating, variable))
comedy_combo_plot <- ggplot(data = comedy_combo, aes(x=variable, y=rating)) + geom_boxplot(aes(fill=as.

comedy_combo_plot
```

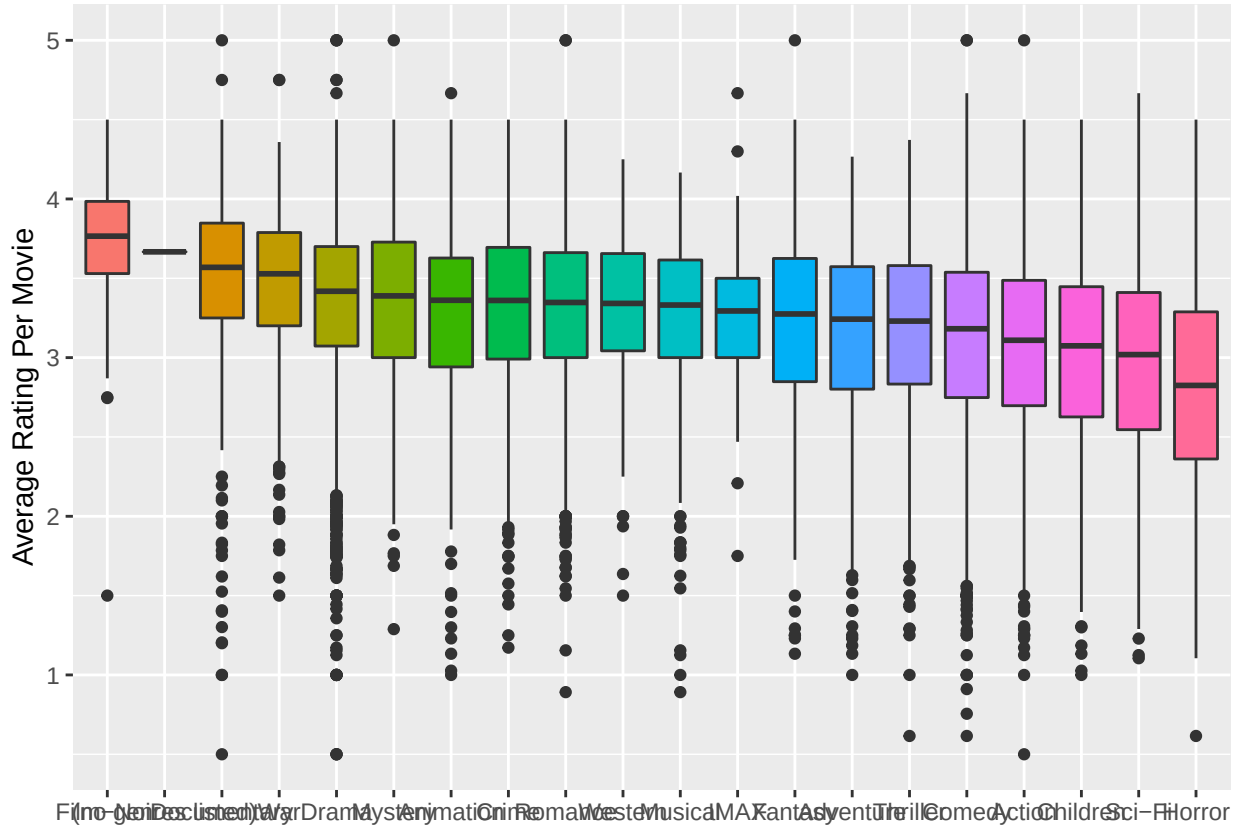


This is exactly what we expected. To further the analysis, I wanted to give voice to my own initial fear and my instinct to avoid something that is essentially an 'Average of Averages' or Simpson's paradox issue. I believe that this is is avoided and the manipulated can be accepted as a proxy in this instance because we are taking many measurements of the same movie instead of masking a collection of underlying values. Additionally, because I expect that the movie averages are significant enough to be in our end model, any effect we lose from this compression will be visible with that feature and accounted for in our model estimation.

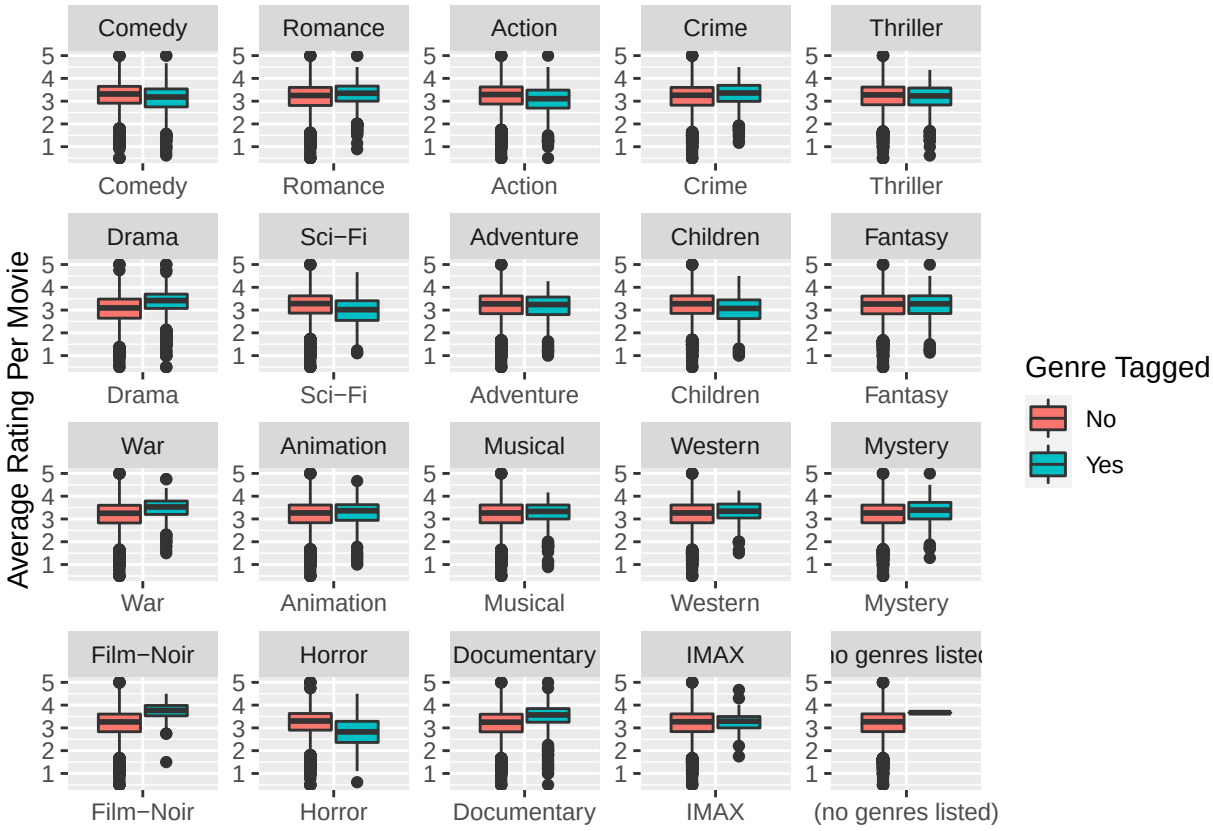
With that in mind and as an acceptable risk, we will being the exploration all over again and attempt the glm fit again. I was excited about this but the results timed out as my computer still was not fast enough.

```
genres_train_set_x <- select(genres_train_set, -userId, -releaseyear, -rating, -reviewday, -reviewyear,
glm <- train(genres_train_set_x,movie_avgs$b_i, method="glm")
# Timeout
```

Since the glm fitted model still will not produce results, we should examine the smaller data set graphically again. We will melt the consolidated data, this time using the average rating per movie instead of looking at each individual rating and plot it.



We can finally see the impact of different genres in the ratings! We can also look at each genre and see the differences between when the genre is present vs when the genre is not:



It seems that there are some genres where this a visible distinction, even to the point where the interquartile ranges do not overlap. While initially excited, I had to temper my enthusiasm after referencing the genre popularity chart. It seems these effects are strongest when the number of films examined is small. This seems problematic, so we need to continue our deep dive before we make any final decisions about how to include genre as a factor in our model.

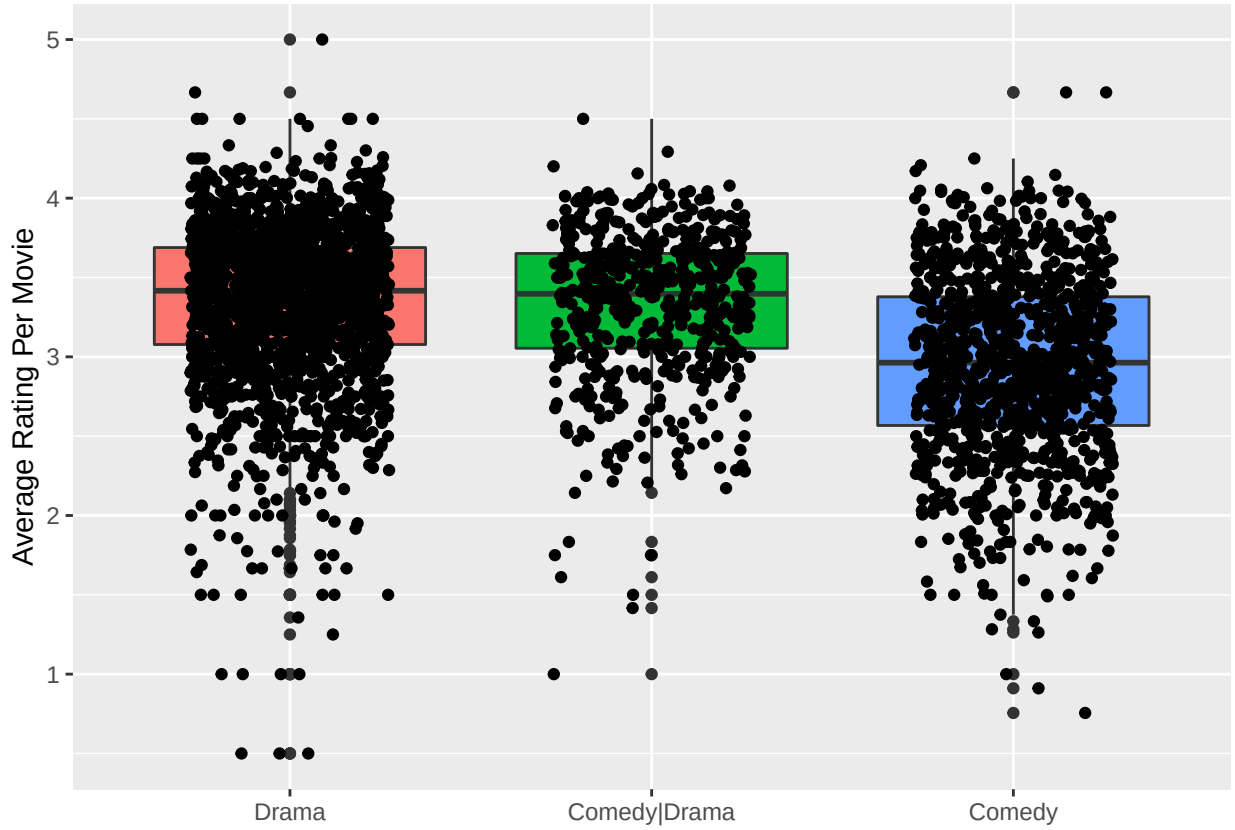
Combo Genre Effects

We have done a lot of work under the assumption that single genres could tell the entire story of a movie's rating better than combined genres, but is that true? We need to find out. We can take a closer look at the original genre list and see which combination of tags are the most common.

```
## # A tibble: 10 x 2
##    genres                      n
##    <chr>                   <int>
##  1 Drama                    1815
##  2 Comedy                   1047
##  3 Comedy|Drama              551
##  4 Drama|Romance             412
##  5 Comedy|Romance            379
##  6 Documentary               350
##  7 Horror                    267
##  8 Comedy|Drama|Romance      255
##  9 Drama|Thriller            192
## 10 Drama|War                 173
```

That really interesting. Both of the top categories are single genres. What happens if we compare the top 2

single genres against the 3rd most popular, which is (conveniently) a 2 genre combination with both other components.



Visually, the combo does not seem to be approximated by the other two individually. This suggests that the combination is more important than any single element and would provide a best estimated fit for our model.

Genre Factor selection

Based on the above analysis, we should include the combination of genres as a factor in our analysis.

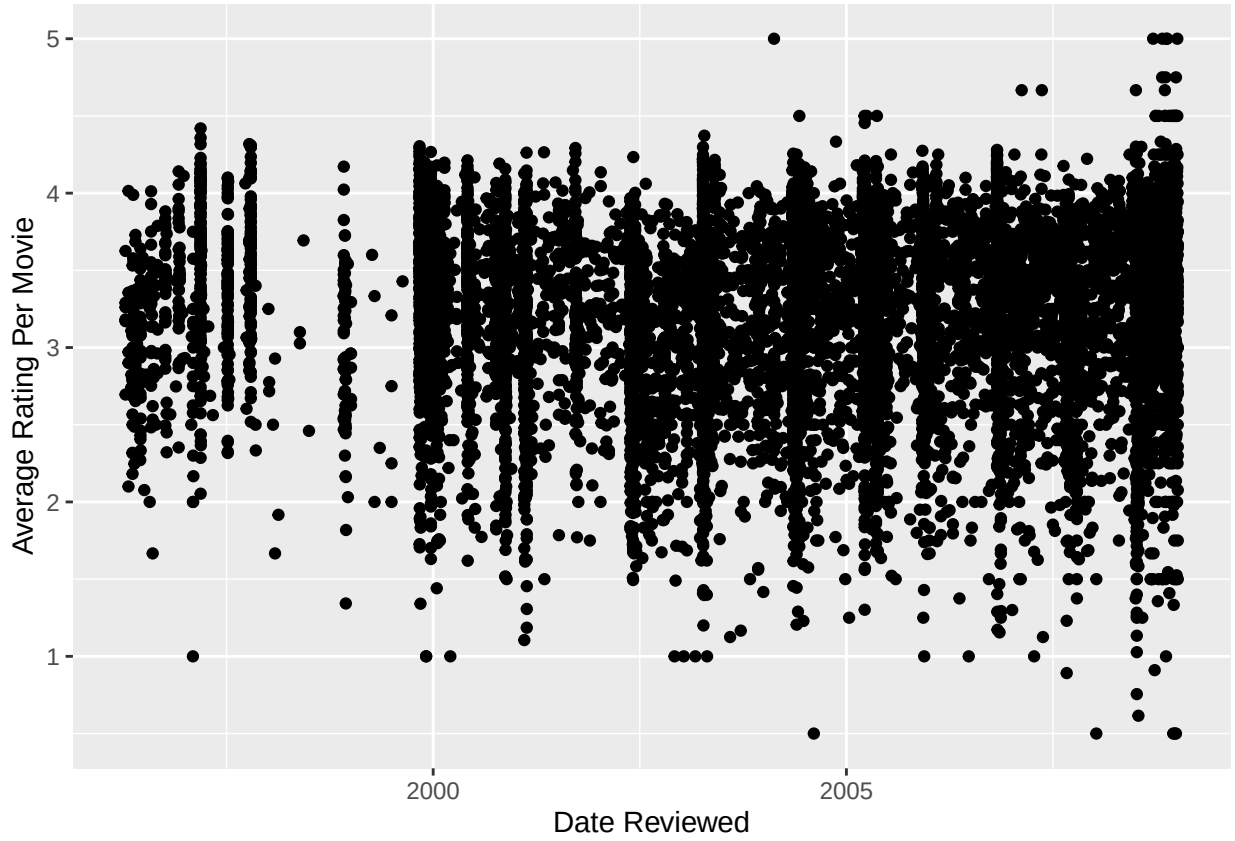
DATES

As we begin with dates, we will simply our data and remove all the genre related columns so we can focus on dates.

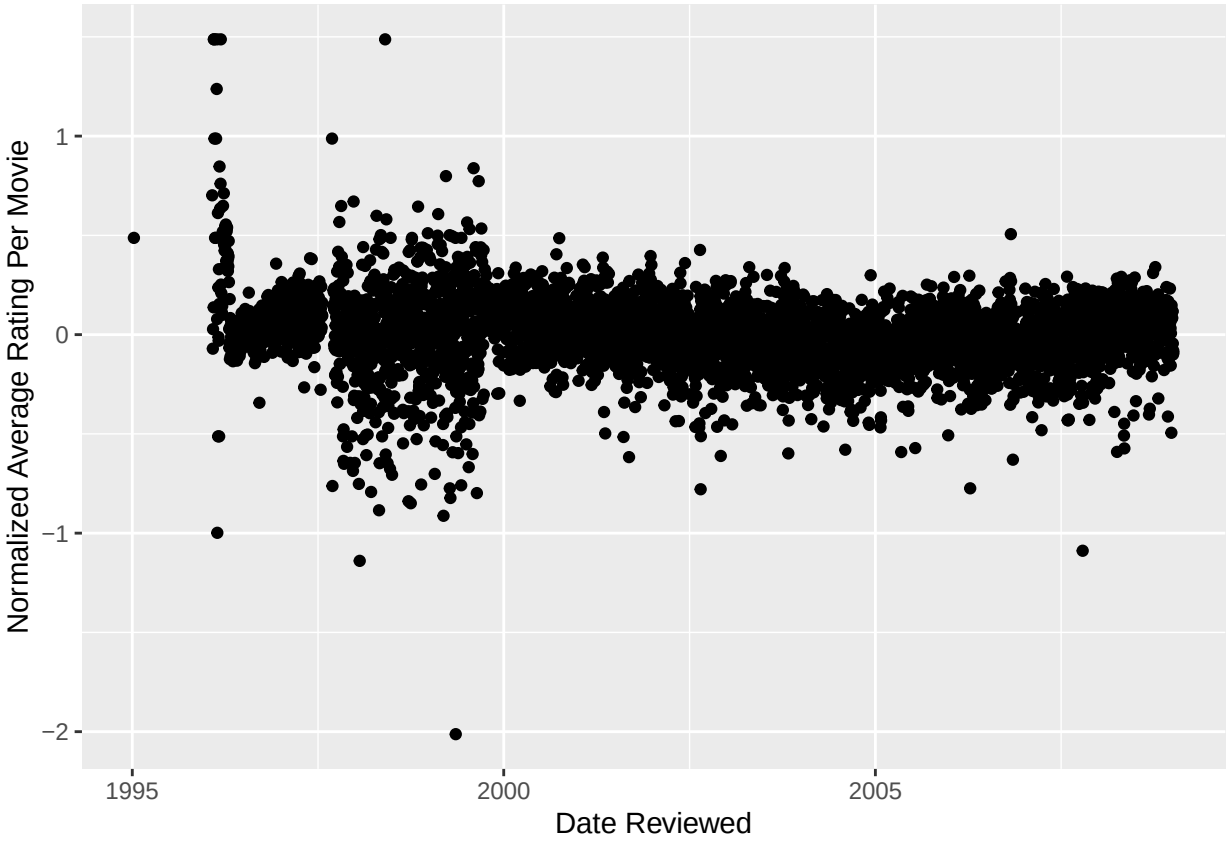
```
date_train_set <- select(tidy_train_set, movieId, userId, rating, releaseyear, reviewday, reviewyear, r
```

The date analysis will be a lot more straightforward and we need to begin by just looking at each of the date elements we have gathered to see if they have noticeable effect on the rating.

Review Date

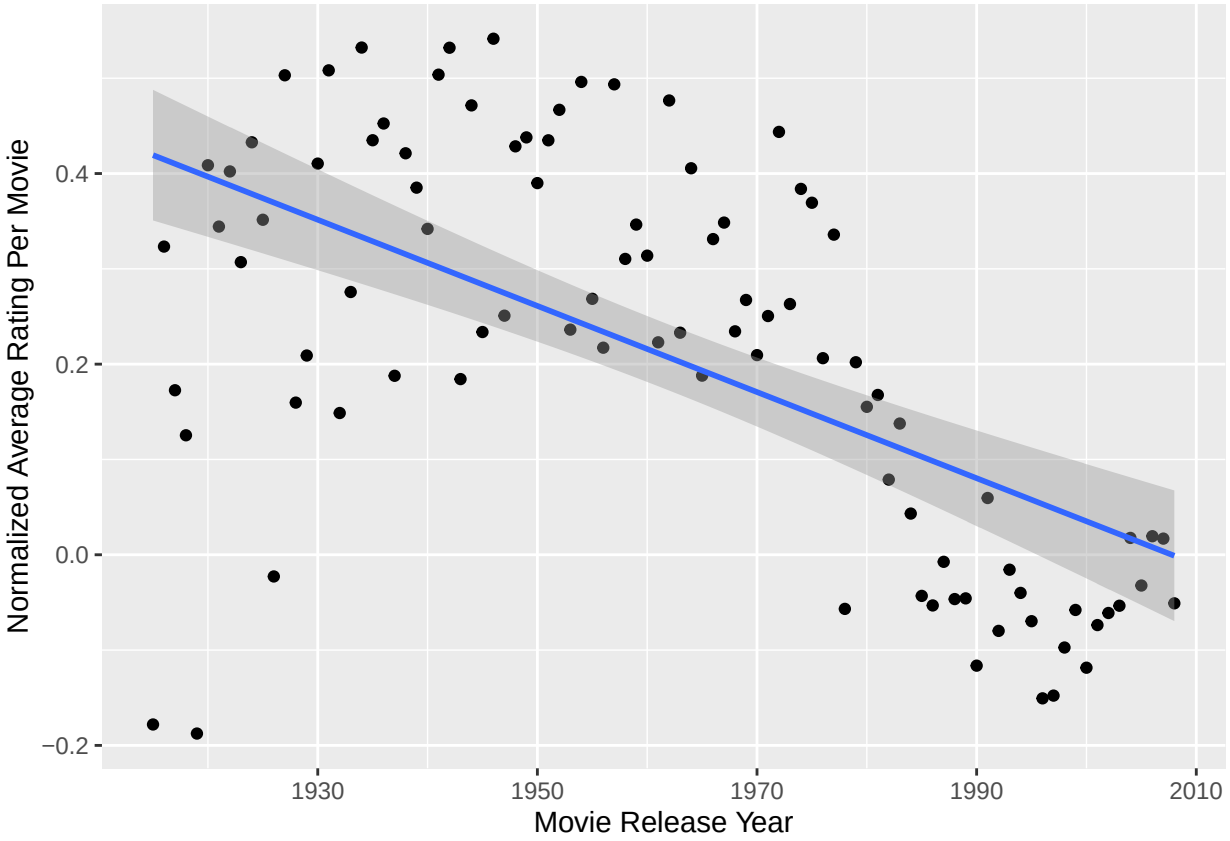


There is no obvious trend here and I did not add a trend line. It seems logical that movie tastes would vary over time, but there is not a visible pattern. We need to normalize the data and see how the ratings deviate from the mean ratings, Maybe this will allow us to see the trend in a different way. This will give us a better perspective and should highlight any insights we are missing.



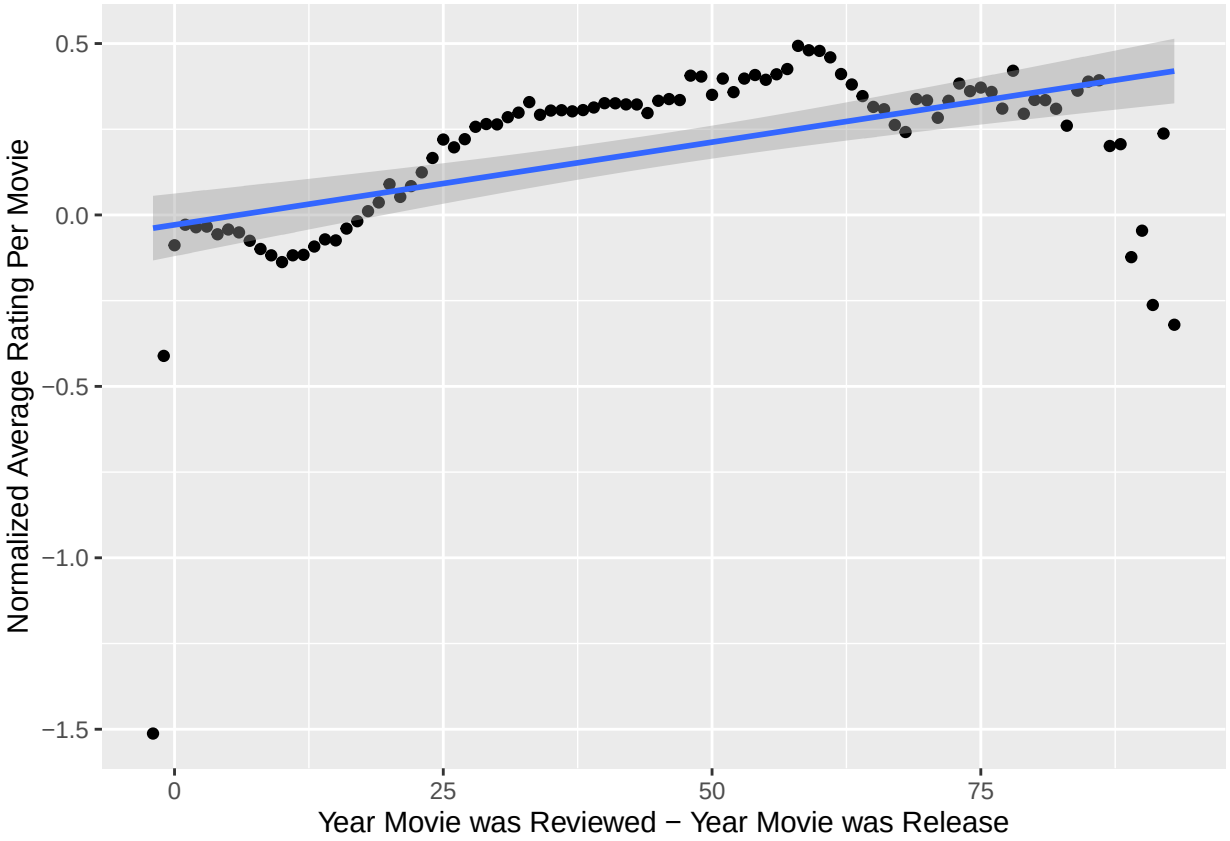
Even with this new perspective the impact of the date the movie was reviewed on does not seem to impact the rating. We will investigate other date factors.

Movie Release Year



There is a much more prominent pattern with with this data. While the fit would likely be better served with a non-linear approximation, the linear model does trend in a meaningful way. This is a good candidate for our model.

Time Between Release and Review



There is another strong correlation here. There are some outliers (like movies that seem to have been reviewed before they were released which hints at messy data) but the linear fit looks excellent, and we should examine this in our model.

Date Factor selection

Based on the above analysis, we should include the movie release year as well as the time between release and review in our model.

Create Models

RMSE - Evaluation Metric and methodology

We are now ready to start putting our models together. We will build each model using only the training data partitioned above. After training, we will generate predictions for each movie in the test partition and test the fit using the root mean square error.

We looked at the math of the RMSE equation above and because we will be using it to evaluation each iteration of our model, we should define a function in R so that we can easily apply it:

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

Once we've tested the different models, we will select the best and use that model to make prediction based on the course **validation** set and see if we achieved our goal.

Baseline

Every project needs to start somewhere, and this one is no different. We'll first take the simplest of all linear models with no features selected and see what kind of fit we can achieve with only the mean.

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

then we test to see what our baseline RMSE is:

```
##          method     RMSE
## 1 Training mean 1.061243
```

We have a long way to go to get to our desired accuracy!

Single Factor Linear Model

We also previously examined the math behind this model, but now we need to convert that model in R. We will use the bias of the individual user as an example and will then look at the individual effect of each of the 5 factors we selected in our investigation. The single factor code will be essentially identical in every model, only varying because we are grouping by and assigning adjustments based on the specific factor we choose to investigate each iteration.

```
user_avgs <- tidy_train_set %>% group_by(userId) %>% summarize(b_u = mean(rating - mu))
predicted_ratings <- mu + tidy_test_set %>% left_join(user_avgs, by='userId') %>% pull(b_u)
```

This model will then be evaluated using the RMSE function above.

Regularized Single Factor Model

Utilizing the same user bias factor again as an example, we will build the regularized single factor model in R. We will start by creating a sequence of λs. This is the sequence we will use for every evaluation. Then we will use the training data to create the model and make predictions against the test data. We will measure the RMSE at each value of λ and then select the value that minimizes the overall error. We will run this model for each of our individual factors and report the results as well.

```
lambdas <- seq(0, 10, 0.25)
sum <- tidy_train_set %>% group_by(userId) %>% summarize(s = sum(rating - mu), n_u = n())
rmses <- sapply(lambdas, function(lu){
     predicted_ratings <- tidy_test_set %>%
          left_join(sum, by='userId') %>%
          mutate(e_u = s/(n_u+lu)) %>%
          mutate(pred = mu + e_u) %>%
          .$pred
     return(RMSE(predicted_ratings, tidy_test_set$rating))
})
regularized_w_user_bias <- rmses[which.min(rmses)]
```

Multiple Factor Linear Models

As we discussed in the model section, considering multiple factors requires a step-wise approach where a single factor is built out first and is then considered when modeling out the next factor. We'll start here by determining the movie effect:

```
movie_b_i <- tidy_train_set %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))
```

and then we will add that a prediction back into the data. This lets us take that into consideration when we group by our second factor and determine the secondary effect.

```
user_movie_avgs <- tidy_train_set %>% left_join(movie_b_i, by='movieId') %>% group_by(userId) %>% summa
predicted_ratings <- tidy_test_set %>% left_join(movie_b_i, by='movieId') %>% left_join(user_movie_avgs
```

These predictions are fed into the RMSE and evaluated.

```
baseline_w_user_movie_bias <- RMSE(predicted_ratings, tidy_test_set$rating)
rmse_results <- add_row(rmse_results, method = "User + movie bias", RMSE = baseline_w_user_movie_bias)
```

Regularized Multiple Factor Linear Model

The same methodology is used when we regularize the model. We must first model a single factor and then build on that information to measure each subsequent factor's effect. We use the same λ sequence and identify one value of λ for all stages of each iteration. Once the individual factors are calculated, predictions will be made against the test set and final values are selected and recorded.

```
rmses <- sapply(lambdas, function(lnu){
     mu <- mean(tidy_train_set$rating)
     b_i <- tidy_train_set %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+lnu))
     b_u <- tidy_train_set %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+lnu))
     predicted_ratings <-
          tidy_test_set %>%
          left_join(b_i, by = "movieId") %>%
          left_join(b_u, by = "userId") %>%
          mutate(prediction = mu + b_i + b_u) %>%
          .$prediction
     return(RMSE(predicted_ratings, tidy_test_set$rating))
})
regularized_w_user_movie_bias <- rmses[which.min(rmses)]
```

We have shown the two step approach, but as a final model, all factors will be combined into a regularized linear model. We evaluate the results below.

```
# regularized multi bias
rmses <- sapply(lambdas, function(llm){
     mu <- mean(tidy_train_set$rating)
     b_i <- tidy_train_set %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+llm))
     b_u <- tidy_train_set %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+llm))
     d_i <- tidy_train_set %>%
          left_join(b_i, by = "movieId") %>%
          left_join(b_u, by = "userId") %>%
          group_by(reviewdelta) %>%
          summarize(d_i = sum(rating - b_u - b_i - mu)/(n()+llm))
     g_i <- tidy_train_set %>%
          left_join(b_i, by = "movieId") %>%
          left_join(b_u, by = "userId") %>%
```

```
          left_join(d_i, by = "reviewdelta") %>%
          group_by(genres) %>%
          summarize(g_i = sum(rating - d_i - b_u - b_i - mu)/(n()+llm))
     r_i <- tidy_train_set %>%
          left_join(b_i, by = "movieId") %>%
          left_join(b_u, by = "userId") %>%
          left_join(d_i, by = "reviewdelta") %>%
          left_join(g_i, by = "genres") %>%
          group_by(releaseyear) %>%
          summarize(r_i = sum(rating - g_i - d_i - b_u - b_i - mu)/(n()+llm))
     predicted_ratings <-
          tidy_test_set %>%
          left_join(b_i, by = "movieId") %>%
          left_join(b_u, by = "userId") %>%
          left_join(d_i, by = "reviewdelta") %>%
          left_join(g_i, by = "genres") %>%
          left_join(r_i, by = "releaseyear") %>%
          mutate(prediction = mu + b_i + b_u + d_i + g_i + r_i) %>%
          .$prediction
     return(RMSE(predicted_ratings, tidy_test_set$rating))
})
regularized_multi_bias <- rmses[which.min(rmses)]
rmse_results <- add_row(rmse_results, method = "Regularized 5 factor bias", RMSE = regularized_multi_bia
```

Evaluation

Training Results

```
##                            method      RMSE
## 1                  Training mean 1.0612426
## 2                      User bias 0.9798774
## 3           Regularized user bias 0.9792256
## 4                     Movie bias 0.9441895
## 5          Regularized movie bias 0.9441261
## 6                Review date bias 1.0524201
## 7    Regularized review date bias 1.0524201
## 8                     Genre bias 1.0186799
## 9         Regularized grenre bias 1.0186766
## 10             Release date bias 1.0501454
## 11 Regularized release date bias 1.0501454
## 12             User + movie bias 0.8667745
## 13 Regularized user + movie bias 0.8660529
## 14     Regularized 5 factor bias 0.8651406
```

It is clear that regularization models are more accurate than the plain linear models, we we expected. Additionally, the more factors that are added the better the results are, again as predicted. Additionally, it does not appear that we are at a point where the data is overly fit so I believe the final regularized model using all of the factors selected will achieve the required result against the validation set.

Validation Results

```
rmses <- sapply(lambdas, function(llm){
     mu <- mean(tidy_train_set$rating)
```

```
    b_i <- tidy_train_set %>%
        group_by(movieId) %>%
        summarize(b_i = sum(rating - mu)/(n()+llm))
    b_u <- tidy_train_set %>%
        left_join(b_i, by="movieId") %>%
        group_by(userId) %>%
        summarize(b_u = sum(rating - b_i - mu)/(n()+llm))
    d_i <- tidy_train_set %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        group_by(reviewdelta) %>%
        summarize(d_i = sum(rating - b_u - b_i - mu)/(n()+llm))
    g_i <- tidy_train_set %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        left_join(d_i, by = "reviewdelta") %>%
        group_by(genres) %>%
        summarize(g_i = sum(rating - d_i - b_u - b_i - mu)/(n()+llm))
    r_i <- tidy_train_set %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        left_join(d_i, by = "reviewdelta") %>%
        left_join(g_i, by = "genres") %>%
        group_by(releaseyear) %>%
        summarize(r_i = sum(rating - g_i - d_i - b_u - b_i - mu)/(n()+llm))
    predicted_ratings <-
        tidy_validation_set %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        left_join(d_i, by = "reviewdelta") %>%
        left_join(g_i, by = "genres") %>%
        left_join(r_i, by = "releaseyear") %>%
        mutate(prediction = mu + b_i + b_u + d_i + g_i + r_i) %>%
        .$prediction
    return(RMSE(predicted_ratings, tidy_validation_set$rating))
})
regularized_valid_bias <- rmses[which.min(rmses)]
rmse_results <- add_row(rmse_results, method = "Validation", RMSE = regularized_valid_bias)
```

```
##       method      RMSE
## 1 Validation 0.8647801
```

We have achieved our desired result!

Results

We created a movie recommender system based on the MovieLens data set with a fairly good accuracy. We imported the raw data, manipulated it and used the available information to select factors that would help us build regression models. We selected two different model types, Linear and Regularized Linear and created several models. These models were evaluated using a root mean squared error and the most successful was selected. This final model was run against a validation set provided and was accurate with a RMSE below 0.86490.