MovieLens Project Report

Kayode Taiwo

10/1/2019

INTRODUCTION

The movielens project aims to create a movie recommendation system using the 10M version of the movielens dataset. Recommendation systems use ratings that *users* have given *items* to make specific recommendations. First we will partition the dataset into a training set and a validation set, with validation comprising 10% of the entire dataset. Second, we will train a machine learning algorithm using the inputs in the training subset to predict movie ratings in the validation set. This introdcution section will describe our dataset, summarize the goal of the project and key steps that were performed.

The dataset

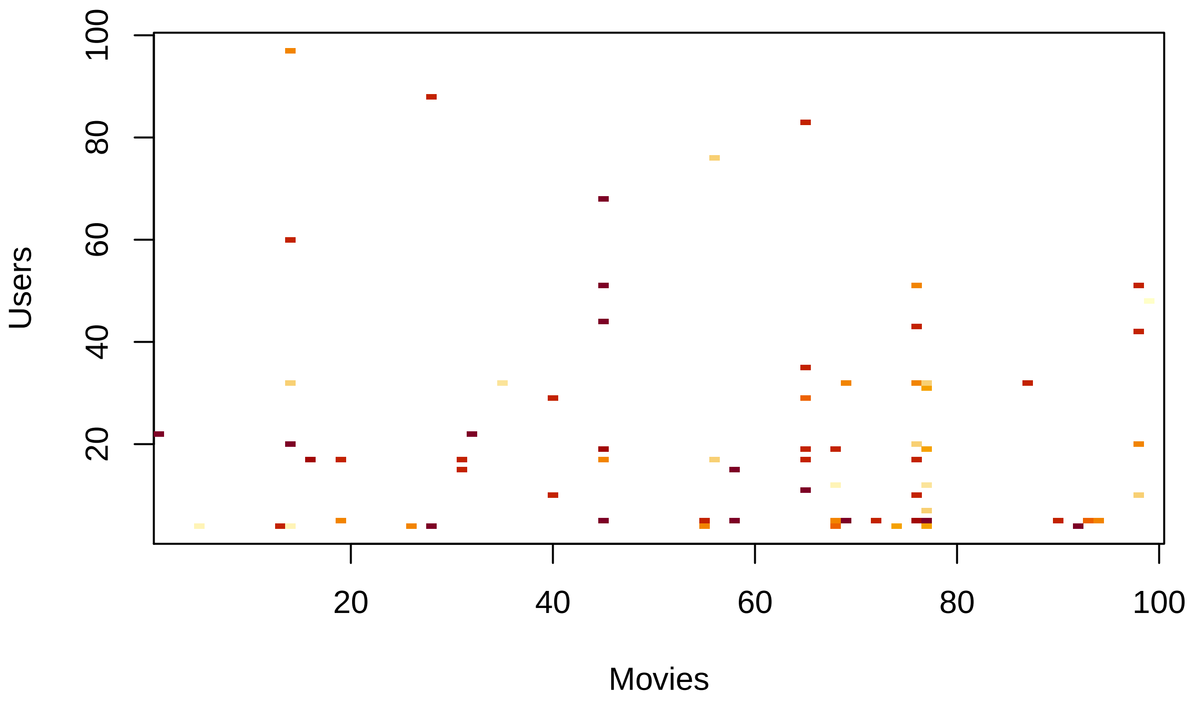
The training set has 9000055 rows and 6 columns. The table below shows the first 10 rows of our training set. Each row represents a rating given by one user to one movie.

```
## # A tibble: 9,000,055 x 6
##    userId movieId rating timestamp title              genres
##     <int>   <dbl>  <dbl>     <int> <chr>              <chr>
##  1      1     122      5 838985046 Boomerang (1992)   Comedy|Romance
##  2      1     185      5 838983525 Net, The (1995)    Action|Crime|Thrill~
##  3      1     292      5 838983421 Outbreak (1995)    Action|Drama|Sci-Fi~
##  4      1     316      5 838983392 Stargate (1994)    Action|Adventure|Sc~
##  5      1     329      5 838983392 Star Trek: Generat~ Action|Adventure|Dr~
##  6      1     355      5 838984474 Flintstones, The (~ Children|Comedy|Fan~
##  7      1     356      5 838983653 Forrest Gump (1994) Comedy|Drama|Romanc~
##  8      1     362      5 838984885 Jungle Book, The (~ Adventure|Children|~
##  9      1     364      5 838983707 Lion King, The (19~ Adventure|Animation~
## 10      1     370      5 838984596 Naked Gun 33 1/3: ~ Action|Comedy
## # ... with 9,000,045 more rows
```

The training set contains the ratings provided by 69878 unique users for 10677 unique movies. This data can be reshaped as a *user-item* matrix with users on the rows and movies on the columns and dimensions 69878 x 10677. The number of rows is equal to the number of unique users, the number of columns is equal to the number of unique movies and the elements are the numeric ratings provided by each user for each movie. An element value of NA means a user has not provided a rating for a movie.

```
##     Toy Story (1995) Jumanji (1995) Grumpier Old Men (1995)
## 175                4             NA                      NA
## 176               NA             NA                      NA
## 177                4             NA                      NA
## 178                4             NA                      NA
## 179               NA             NA                      NA
## 180               NA              4                      NA
## 182                4              3                       3
## 183               NA              4                       5
## 184               NA              2                      NA
## 185               NA             NA                      NA
## 186               NA             NA                      NA
```

An intuition of the **sparsity** of the *user-item* matrix can be seen by plotting an image of the matrix for a random sample of 100 movies and 100 users with a colored dot indicating a user/movie combination for which we have a rating.



The five most given ratings in order from most to least are: 4, 3, 5, 3.5, 2.

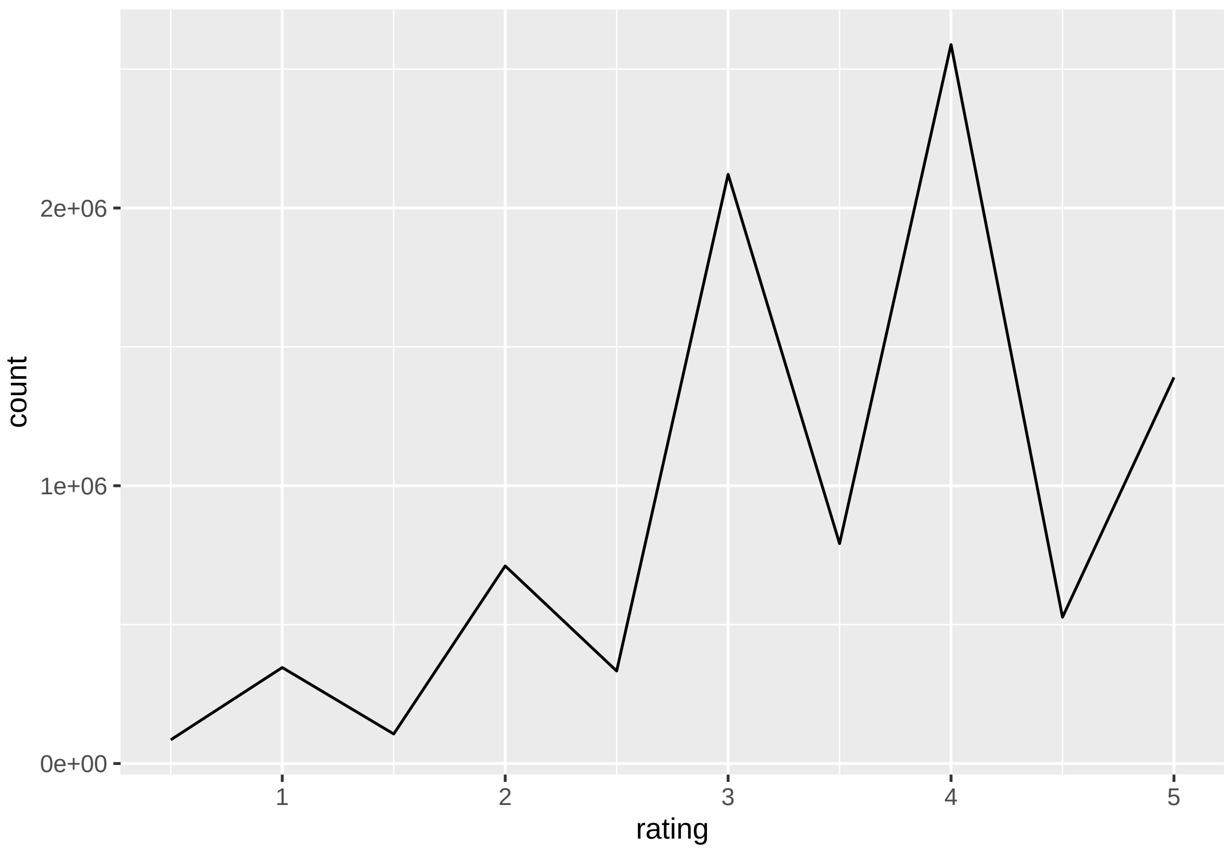
```
## Selecting by count
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

The movie with the highest number of ratings is Pulp Fiction. The table below shows the top 10 movies by number of ratings ordered in descending order of number of ratings.

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                    count
##      <dbl> <chr>                                                    <int>
##  1     296 Pulp Fiction (1994)                                      31362
##  2     356 Forrest Gump (1994)                                      31079
##  3     593 Silence of the Lambs, The (1991)                         30382
##  4     480 Jurassic Park (1993)                                     29360
##  5     318 Shawshank Redemption, The (1994)                         28015
##  6     110 Braveheart (1995)                                        26212
##  7     457 Fugitive, The (1993)                                     25998
##  8     589 Terminator 2: Judgment Day (1991)                        25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (19~ 25672
## 10     150 Apollo 13 (1995)                                         24284
## # ... with 10,667 more rows
```

Movies are rated from 0.5 to 5.0 in 0.5 increments, with full star ratings (1, 2, 3, 4, 5) more commonplace than half star ratings (0.5, 1.5, 2.5, 3.5, 4.5).



Movies in the dataset are grouped by genre (drama, comedy, action, etc) with each movie belonging to zero, one or more genres.

```
## # A tibble: 20 x 2
##    genres              count
##    <chr>               <int>
##  1 Drama             3910127
##  2 Comedy            3540930
##  3 Action            2560545
##  4 Thriller          2325899
##  5 Adventure         1908892
##  6 Romance           1712100
##  7 Sci-Fi            1341183
##  8 Crime             1327715
##  9 Fantasy            925637
## 10 Children           737994
## 11 Horror             691485
## 12 Mystery            568332
## 13 War                511147
## 14 Animation          467168
## 15 Musical            433080
## 16 Western            189394
## 17 Film-Noir          118541
## 18 Documentary          93066
## 19 IMAX                  8181
## 20 (no genres listed)       7
```

The goal

The goal of this project is to train a machine learning model with data from the training set and predict movie ratings with data from the validation set. We also seek to measure model performance by applying the residual mean square error (RMSE) loss function.

The steps performed

The following steps were performed in order to realise this project:

- Data partitioning. 10% validation, rest training
- Model selection and definition
- Loss function selection and definition
- Estimating global average rating ($\hat{\mu}$)
- Regularization and Penalized Regression for movie ($\hat{b}_i(\lambda)$) and user ($\hat{b}_u(\lambda)$) effects
- Calculation of regularization penalty (λ) using cross-validation
- Estimating regularized average rating per movie
- Estimating regularized average rating per user
- Estimating residuals using Matrix factorization

METHODS

Our model

We model the rating a user u will give a movie i with the following model:

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

where

- μ = average rating across all users and all movies,
- b_i = regularized average rating per movie - average of $Y_{u,i} - \mu$ for each movie i,
- b_u = regularized average rating per user - average of $Y_{u,i} - \mu - b_i$ for each user u,
- $r_{u,i}$ = residuals after normalizing $Y_{u,i}$. That is $Y_{u,i} - \mu - b_i - b_u$ for each movie i and each user u,
- $\varepsilon_{u,i}$ = independent errors sampled from the same distribution centered at 0.

We obtain estimates for each of the terms in our model as listed above using the training dataset.

Loss function

Our loss function is based on the residual mean square error (RMSE) and is calculated on the validation dataset. We define $y_{u,i}$ as the rating for movie i by user u and denote our prediction with $\hat{y}_{u,i}$. The RMSE is then defined as:

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

where N is the number of user/movie combinations and the sum is over all user/movie combinations.

It should be noted that we can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star, which is not good. We define a function that computes the RMSE for vectors of ratings and their corresponding predictors:

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

Estimating global average rating

The estimate for global average rating (μ) that minimizes the RMSE is the least squares estimate of μ and, in this case, is the average of all ratings:

```r
mu <- mean(edx$rating)
mu
```

```
## [1] 3.512465
```

Regularization

Regularization permits us to penalize large estimates that are formed using small sample sizes. Instead of minimizing the least square equation (as defined y our model):

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

we minimize an equation that adds a penalty (λ):

$$\frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2)$$

The values of b_i that minimize this equation are:

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

where η_i is the number of ratings made for movie i.

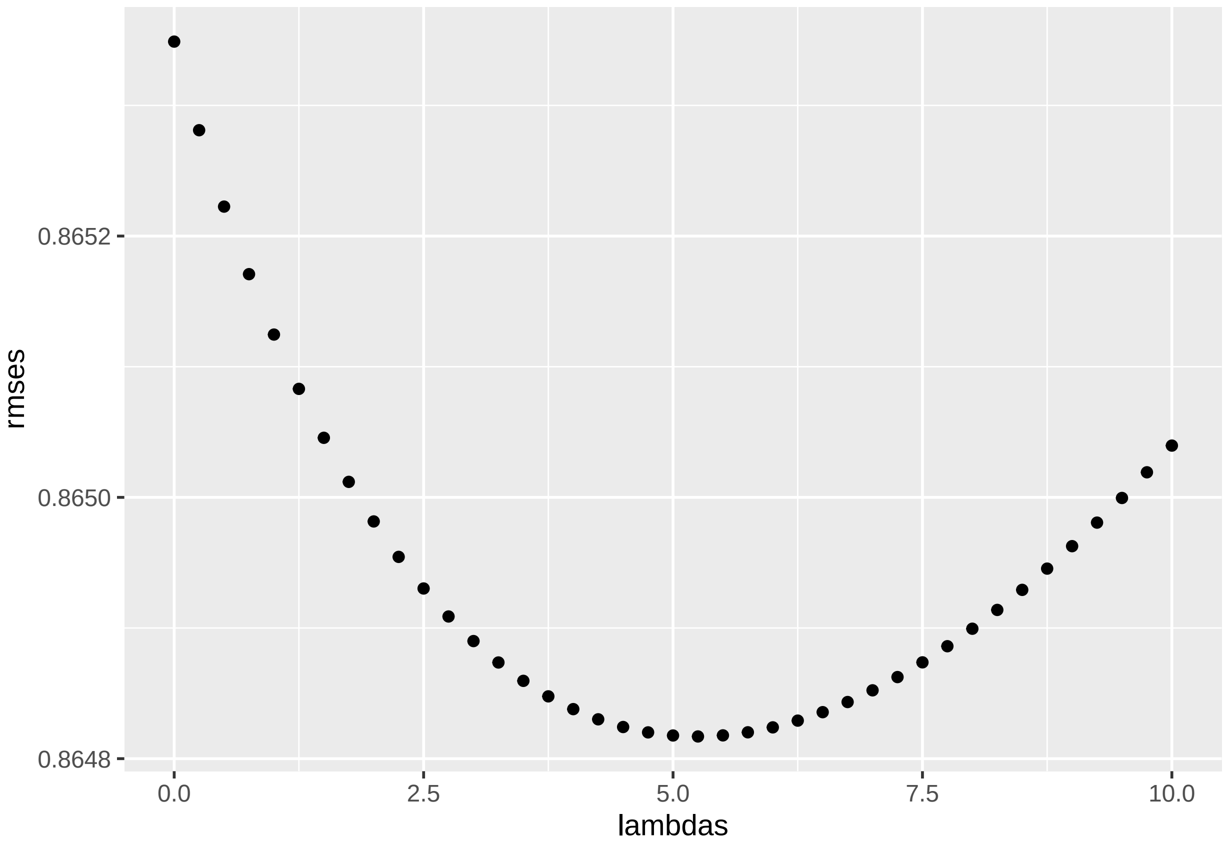
The values of b_u that minimizes this equation are:

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

where η_u is the number of ratings made by user u.

Note that λ is a tuning parameter. We can use **cross-validation** to choose it.

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

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

  mu <- mean(edx$rating)

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

  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

    return(RMSE(predicted_ratings, validation$rating))
})

qplot(lambdas, rmses)
```



The optimal λ can be obtained as:

```
lambda <- lambdas[which.min(rmses)]
lambda
```

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

Estimating regularized average rating per movie

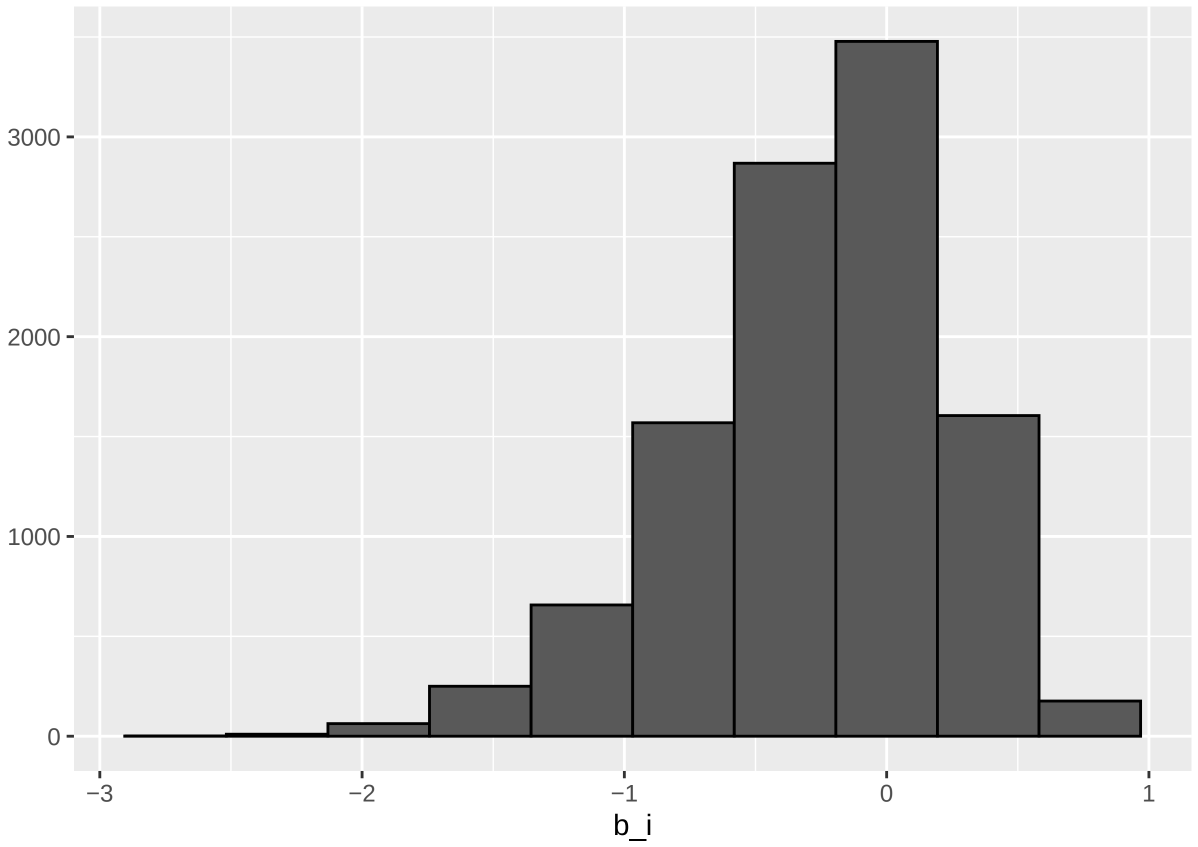
The term, b_i represents the average rating for movie i. The b_is are referred to as "movie effects" or "movie bias". We can use least squares to estimate the b_i like this:

```
fit <- lm(rating ~ as.factor(movieId), data = edx)
```

Because there are thousands of b_i as each movie gets one, the lm() function will be very slow. So we do not use the code above. Instead we compute the least squares estimate $\hat{b}_i$ as the average of $Y_{u,i} - \hat{\mu}$ for each movie i:

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

We can see that these estimates vary substantially due to movie to movie variability (different movies are rated differently):



Estimating regularized average rating per user

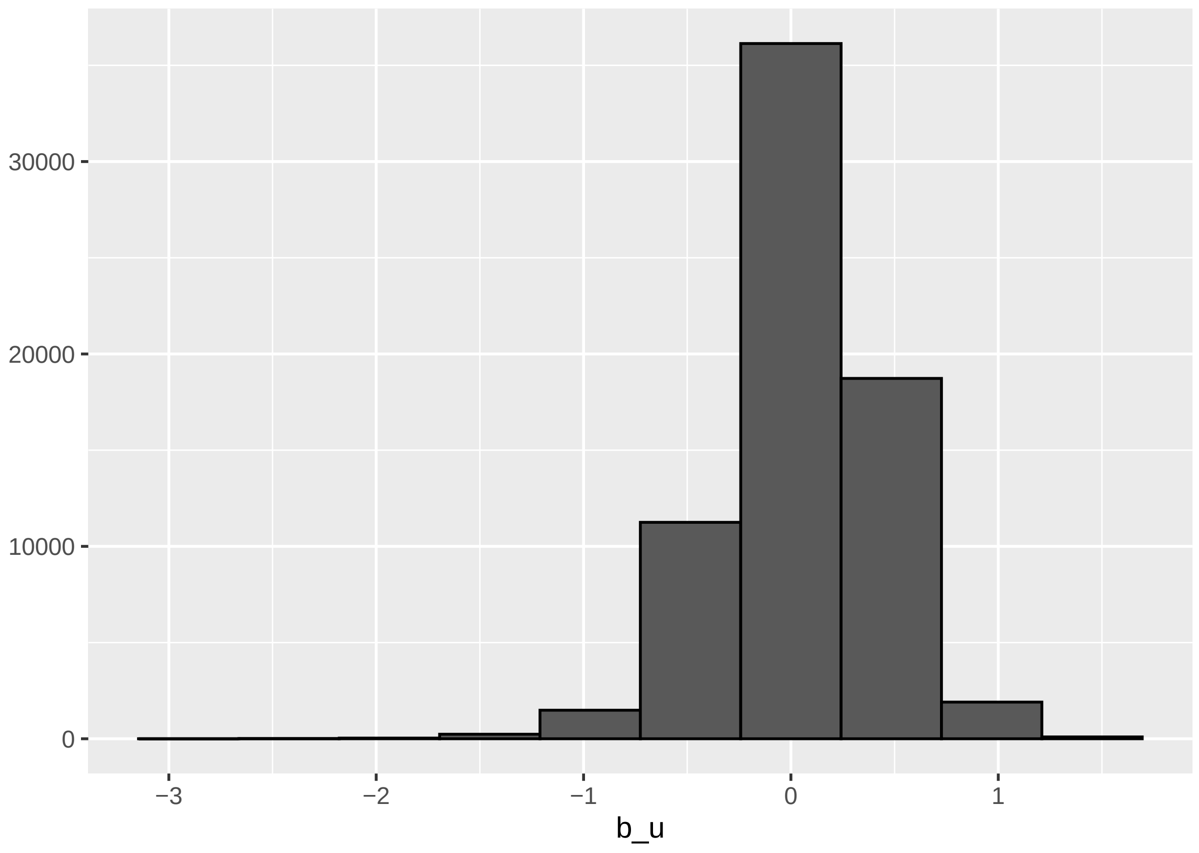
The term, b_u represents the average rating for user u. The b_us are referred to as "user effects" or "user bias". We can use least squares to estimate the b_u like this:

```
lm(rating ~ as.factor(movieId) + as.factor(userId))
```

Because there are thousands of b_u as each user gets one, the lm() function will be very slow. So we do not use the code above. Instead we compute the least squares estimate $\hat{b}_u$ as the average of $Y_{u,i} - \hat{\mu} - b_i$ for each user u:

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

We can see that these estimates vary substantially due to user to user variability (different users rate movies differently):



Estimating residuals using Matrix factorization

Our model:

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

accounts for movie to movie differences through the b_is and user to user differences through the b_us. The residuals ($r_{u,i}$) account for another important source of variation related to the fact that groups of movies have similar rating patterns and groups of users have similar rating patterns as well. We explore these patterns by studying the residuals:

$$Y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u$$

Below again is a small subset of our *user-item* matrix:

```
##     Toy Story (1995) Jumanji (1995) Grumpier Old Men (1995)
## 175                4             NA                      NA
## 176               NA             NA                      NA
## 177                4             NA                      NA
## 178                4             NA                      NA
## 179               NA             NA                      NA
## 180               NA              4                      NA
## 182                4              3                       3
## 183               NA              4                       5
## 184               NA              2                      NA
## 185               NA             NA                      NA
## 186               NA             NA                      NA
```

The full *user-item* matrix has dimensions 69878 x 10677. This huge size has proven to be too large to be used to show processes described here. R crashes every time we try. So we will use a small sub-matrix with dimensions 1000 x 1000 extracted from the full *user-item* matrix as follows.

```r
r <- y[1:1000,500:1500]
rownames(r) <- rownames(y)[1:1000]
colnames(r) <- colnames(y)[500:1500]
```

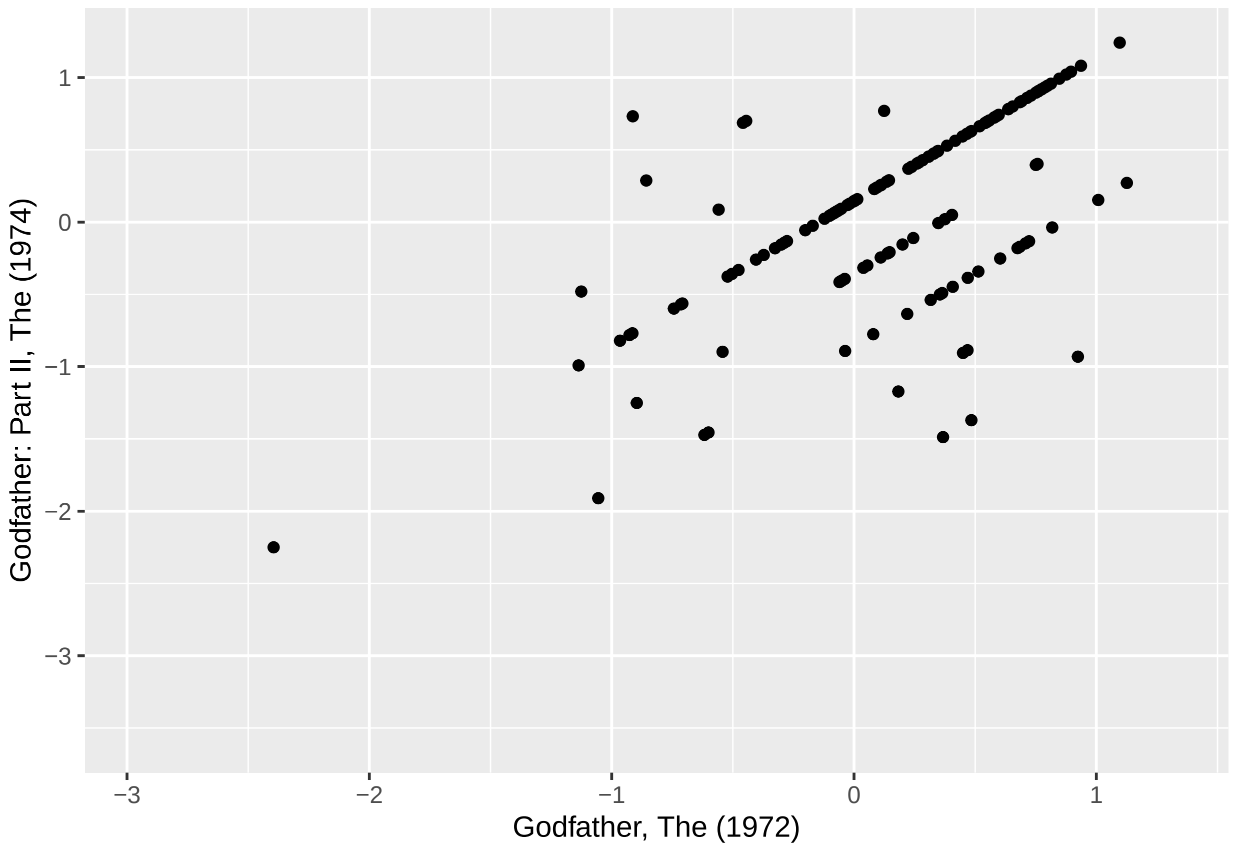
We can convert this to a residuals matrix $(r_{u,i})$ by removing the column and row effects:

```r
r <- sweep(r, 2, colMeans(r, na.rm=TRUE))
r <- sweep(r, 1, rowMeans(r, na.rm=TRUE))
```

We can see that residuals for different movies are not independent from each other by looking at an example using the following code:

```r
m_1 <- "Godfather, The (1972)"
m_2 <- "Godfather: Part II, The (1974)"
qplot(r[,m_1], r[,m_2], xlab = m_1, ylab = m_2)
```

```
## Warning: Removed 857 rows containing missing values (geom_point).
```



By looking at the correlation between movies we deduce that there is structure in the residuals data. We can see very strong correlation patterns if we ran this code on the matrix of residuals:

```r
cor(r)
```

Singular Value Decomposition (SVD) is an algorithm that can create matrices p and q, that can explain much of the structure we see. p is a matrix that has one column for each movie and one row for each factor, while q is a matrix that has one row for each user and one column for each factor. We can write the following math formula for our residuals $r_{u,i}$ with m rows and n columns as:

$$r_{u,i} = p_{u,1}q_{1,i} + p_{u,2}q_{2,i} + ... + p_{u,m}q_{m,i}$$

with the variability of each term decreasing and with the ps uncorrelated. The algorithm also computes this variability so that we can know how much of the matrices total variability is explained as we add new terms. This may permit us to see that, with just a few terms, we can explain most of the variability.

We can compute a singular value decomposition (SVD) on our residuals matrix by using the following code:

```r
s <- svd(r)
```

We cannot use the code above here, because our full residual matrix is very large in this case, with dimensions 69878 x 10677 and R crashes every time we attempted to compute a SVD. But we know however that the SVD is merely a decomposition of the residual matrix that serves to reduce dimensions on the residual matrix. As such we may use our residual matrix directly in our model and we do so using this code:

```r
user_movie_resids <- edx %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  mutate(resid = rating - mu - b_i - b_u) %>%
  select(userId, movieId, resid)
```

RESULTS

In the methods section above we have discussed how we obtain estimates for our model components from the training dataset. In this section we apply the learned model to predict recommendations in the validation dataset. Here is the code that applies the model to the validation dataset:

```r
predicted_ratings <- validation %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  left_join(user_movie_resids, by=c("userId", "movieId")) %>%
  mutate(resid = ifelse(is.na(resid), 0, resid), pred = mu + b_i + b_u + resid) %>%
  pull(pred)
```

To get a sense of how well our model performs we compute a RMSE as follows:

```r
rmse <- RMSE(predicted_ratings, validation$rating)
rmse
```

```
## [1] 0.864817
```

We recall that the RMSE is analogous to the standard deviation of the distribution of the validation dataset it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star, which is not good.

CONCLUSION

By employing a model that accounts for movie effects (b_i), user effects (b_u) and residuals ($r_{u,i}$) we were able to predict movie recommendations with a RMSE less than one. What this means is that we were able to reduce error in each prediction to less than 1 star. While this is sufficient for this project it does have some limitations and can be enhanced in the future as discussed below.

Limitations

Our model does not account for "time effect" or variations in the predicted ratings caused by time. Our model also does not account for "genre effects" or variations in the predicted ratings caused by genres.

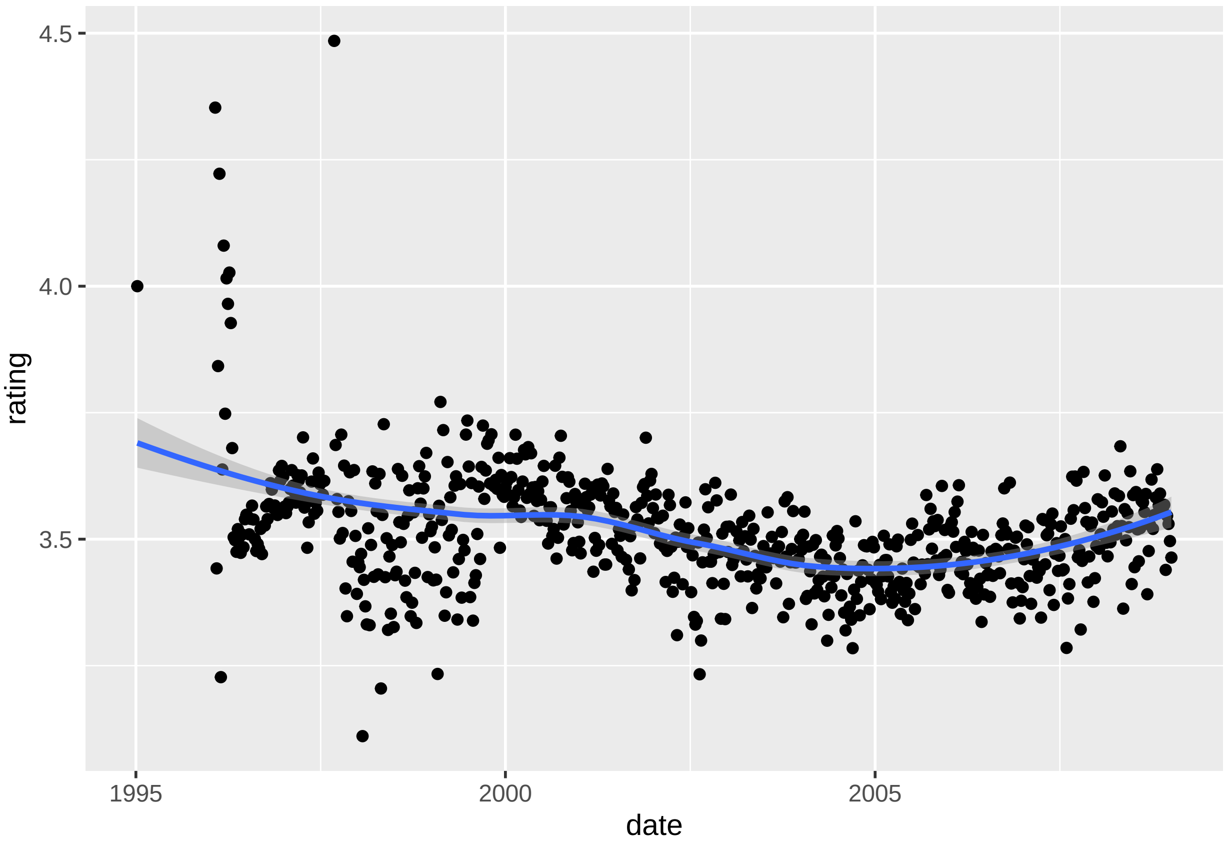
Future work

In a future work on this same project we can achieve even better results by accounting for variations in the dataset due to "time effects" and "genre effects". This means we would enhance our current model to include terms for an estimate of both effects as discussed below.

Modeling time effects

Below is a plot of the average rating for each week plotted against day:

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



The plot shows some evidence of a time effect. If we define $d_{u,i}$ as the day for user's u rating of movie i, then we can enhance our model with a time effect as follows:

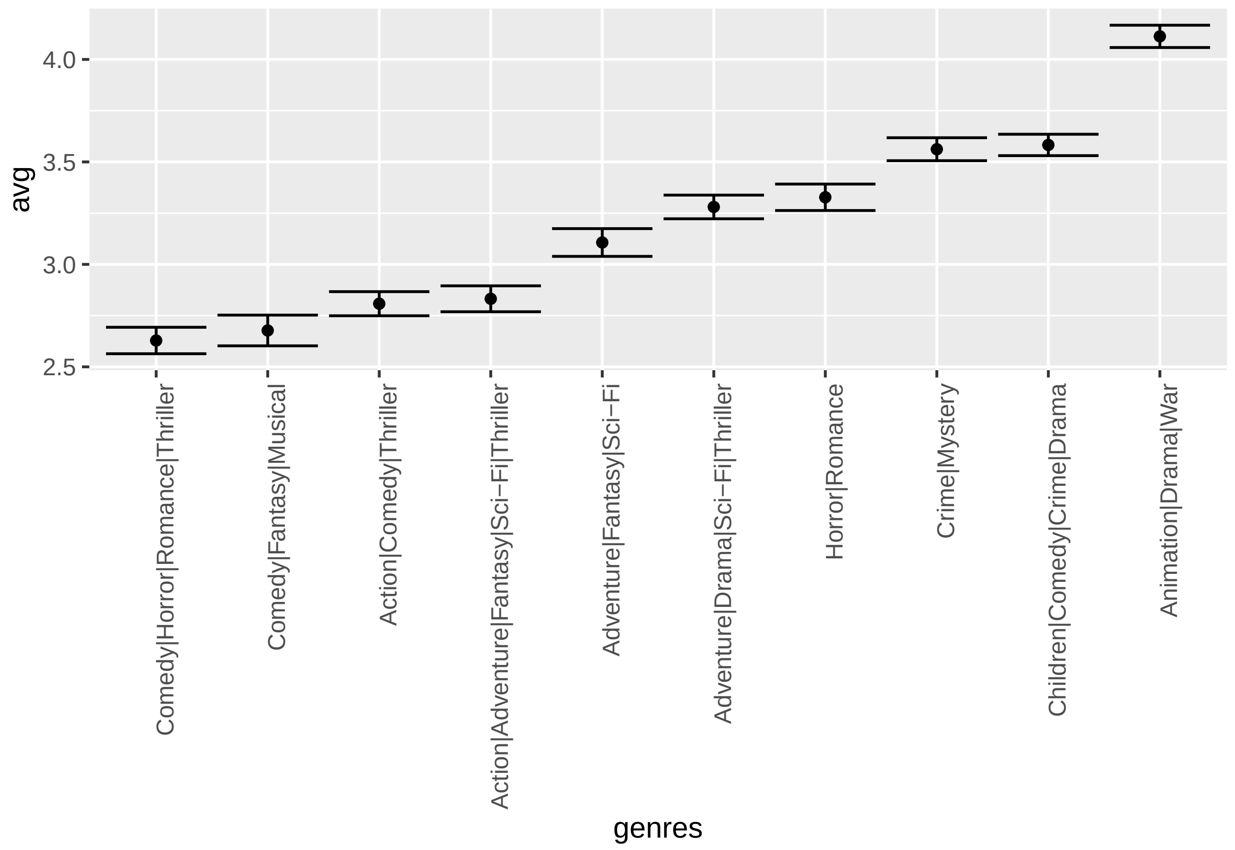
$$Y_{u,i} = \mu + b_i + b_u + r_{u,i} + f(d_{u,i}) + \varepsilon_{u,i}$$

where

f is a smooth function of $d_{u,i}$.

Modeling genre effects

The movielens data also has a genres column. This column includes every genre that applies to the movie. Some movies fall under several genres. If we define a category as whatever combination appears in this column and keep only categories with ratings ranging from 1,000 to 1,100 ratings. Then we compute the average and standard error for each category and plot these as error bar plots.



The plot shows strong evidence of a genre effect. If we define $g_{u,i}$ as the genre for user's u rating of movie i, then we can enhance our model with a genre effect as follows:

$$Y_{u,i} = \mu + b_i + b_u + r_{u,i} + \sum_{k=1}^{K} x_{u,i}\beta_k + \varepsilon_{u,i}$$

with $x_{u,i}^k = 1$ if $g_{u,i}$ is genre k.