MovieLens Project

Yap Kim Thow

1/1/2021

1 INTRODUCTION

In this project, a subset of the MovieLens database is used to create a recommendation system. In this case, to make specific recommendations to users based on ratings which they have given to movies. If a user is predicted to give a five stars rating to a specific movie, then that movie would be a good candidate for a recommendation to be made to the user.

1.1 MovieLens 10M dataset

The data set in this project comprises of a subset of the database of 10 million ratings on about 10,000 movies by 72,000 users, provided by the GroupLens research lab. Snippets of code, given as part of the project assignment, downloads the data set and splits it into an *edx* set with 9,000,055 observations (90%) and a *validation* set with 999,999 observations (10%).

1.1-1 Training set and "pre-validation" set

The project assignment prescribes that a "validation" set being used as the final hold-out test set to calculate the selected model's RMSE, i.e. the *validation* set here (sampled in the previous sub-section) may be more appropriately referred to as the "test" set as per normal convention (as far as I understand it).

For the purpose of model evaluation and selection, I will be using a *pre-validation* set (this would be a "validation" set as per normal convention) sampled from the *edx* set with the remaining observations placed in a *training* set.

1.2 Loss function

The loss function used in this project is the residual mean squared error (RMSE) on a test set:

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

with N being the number of all user-movie combinations, and subscripts u and i denote user and movie, respectively.

The target RMSE value for the recommender system is 0.86490.

The function to calculate RMSE is:

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

2 METHODS/ANALYSIS

This section describes the data exploration steps on the *edx* set, the models considered in this report, the model training and evaluation against the "pre-validation" set, as well as the RMSE value of the selected model tested against the final hold-out set (*validation* set).

2.1 Data exploration

An exploration of the *edx* data set is performed to understand their structure and dimensions.

```r
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
##  - attr(*, ".internal.selfref")=<externalptr>
```

```r
dim(edx)
```

```
## [1] 9000055       6
```

```r
dim(validation)
```

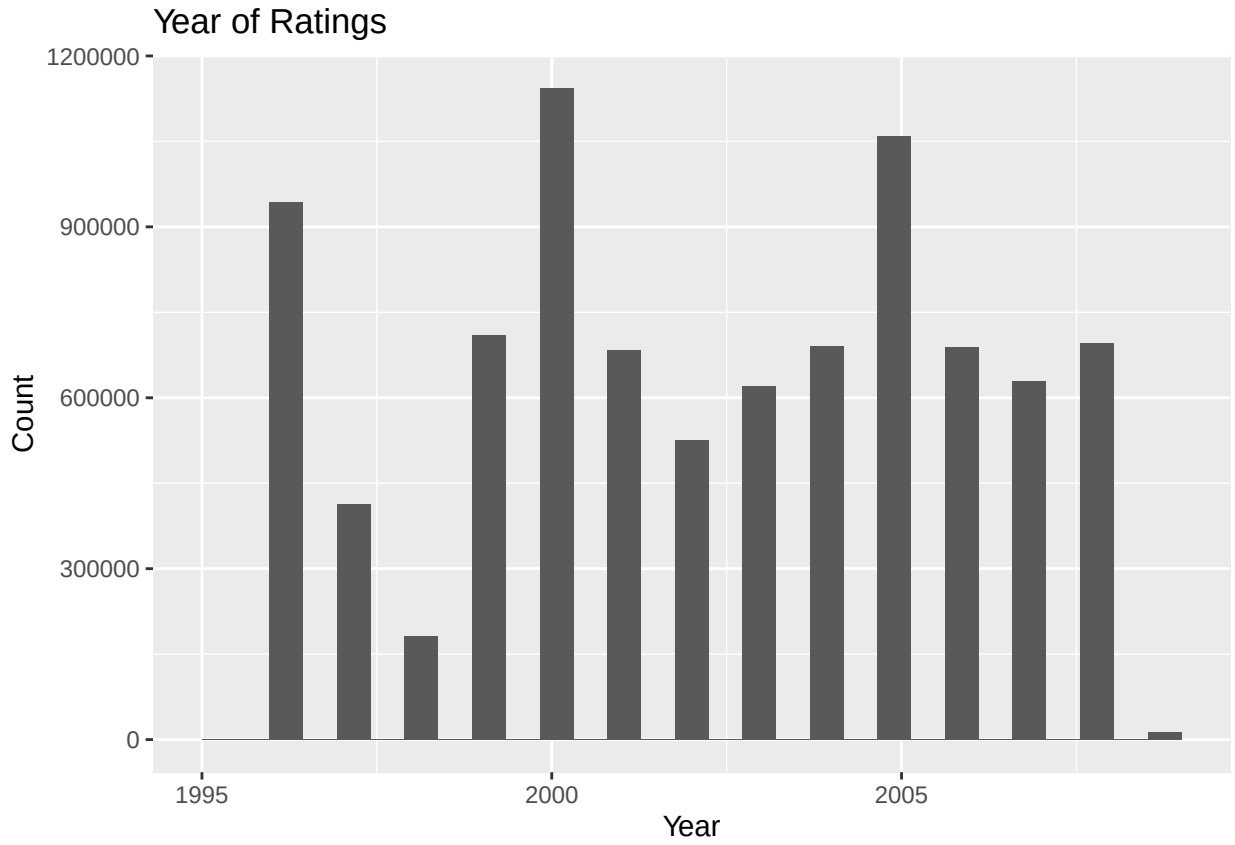
```
## [1] 999999      6
```

```r
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

2.1-1 Date range

The date range of the ratings from which the data sets have data is calculated in the following.

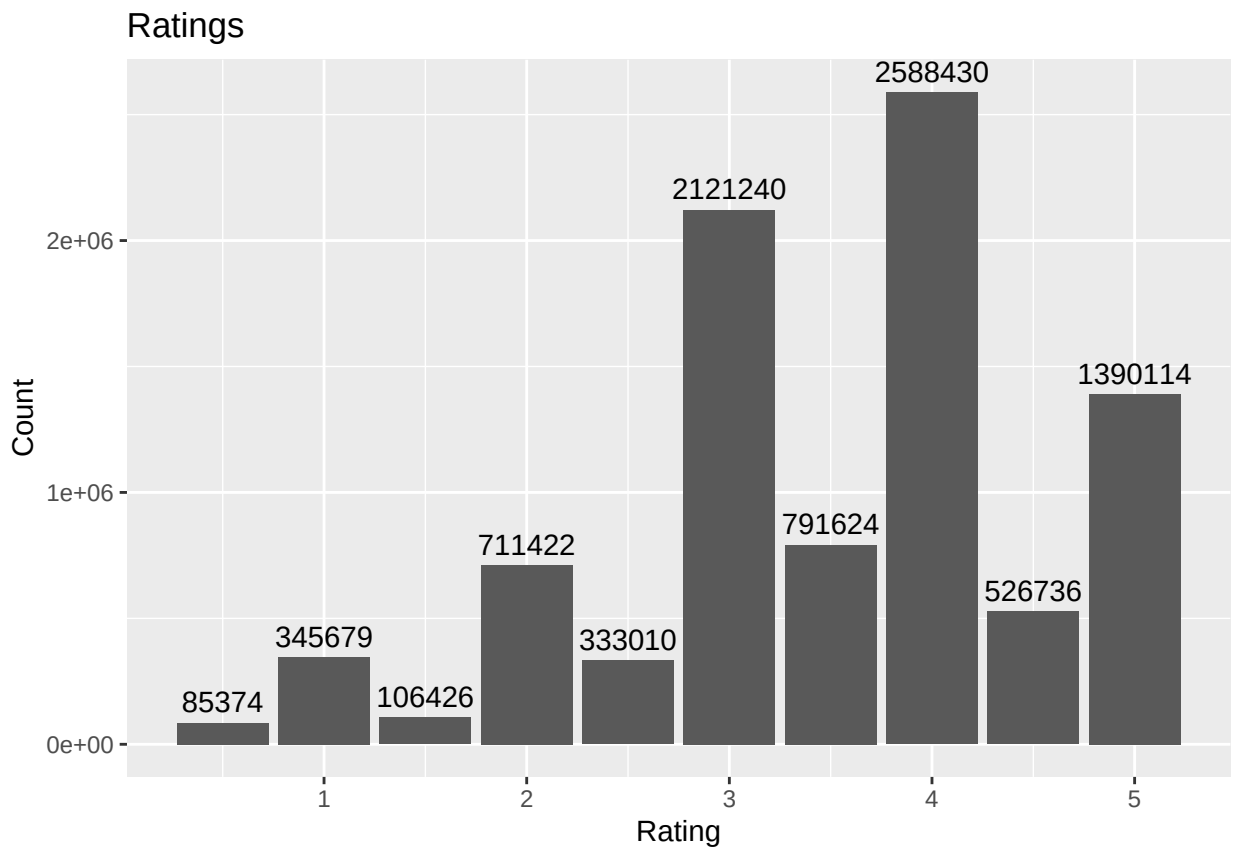


The earliest rating in the *edx* set is dated 1995-01-09 whilst the latest rating is dated 2009-01-05.

2.1-2 Ratings distribution

Next, the distribution of the ratings in the *edx* set is examined.

```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    0.5   85374
##  2    1    345679
##  3    1.5  106426
##  4    2    711422
##  5    2.5  333010
##  6    3   2121240
##  7    3.5  791624
##  8    4   2588430
##  9    4.5  526736
## 10    5   1390114
```



2.1-3 Genres

It would be also useful to examine the number of genres in the *edx* dataset.

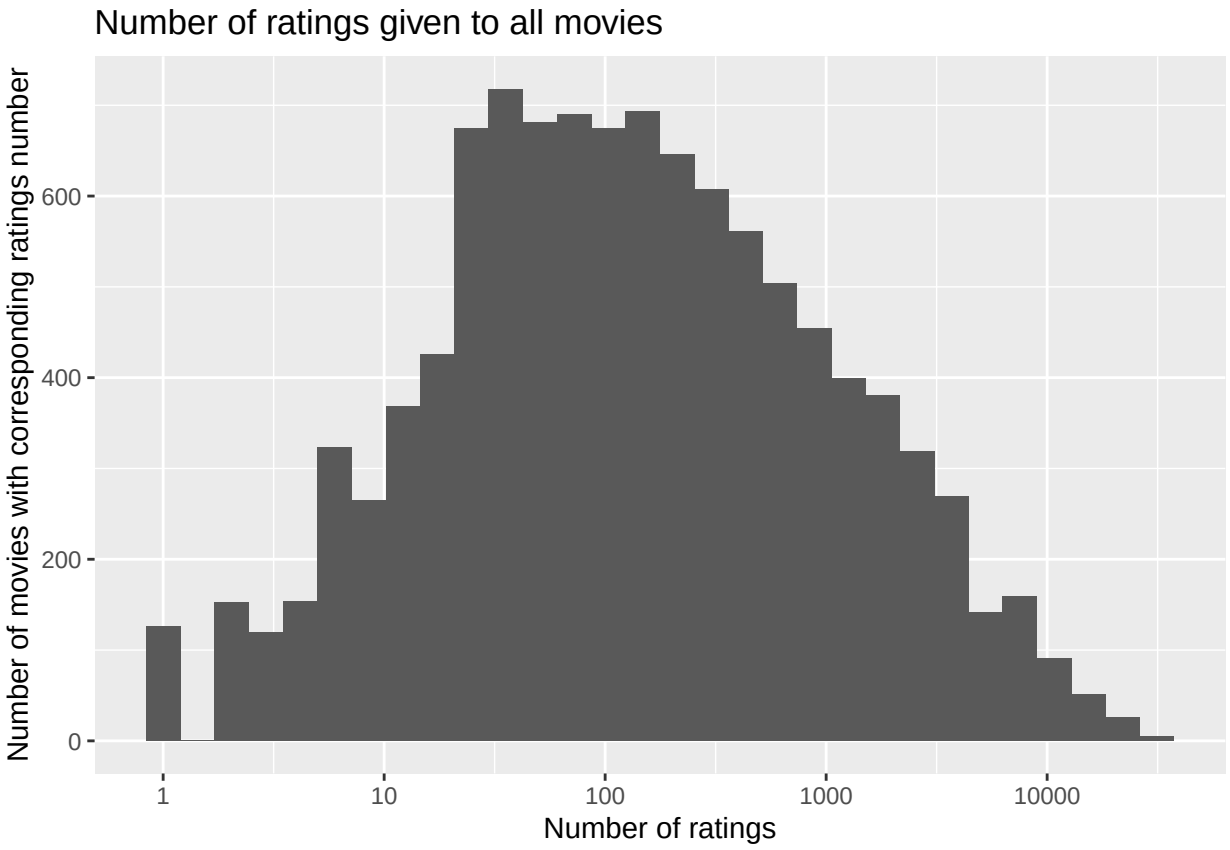
```
## # A tibble: 20 x 2
##    genres                 n
##    <chr>              <int>
##  1 (no genres listed)     7
##  2 Action           2560545
##  3 Adventure        1908892
##  4 Animation         467168
##  5 Children          737994
##  6 Comedy           3540930
##  7 Crime            1327715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

```
## 19 War                 511147
## 20 Western             189394
```

There are a total of 19 distinct genres in the dataset, including a *(no genres listed)* category for 7 ratings.

2.1-4 Number of ratings for movies

It could be useful to examine the number of ratings obtained for all the movies in the data set.



From the above, it can be seen that there are a handful of movies which are/were extremely popular with tens of thousands of ratings. But there are also hundreds of obscure movies with only single ratings or a handful of ratings.

```
## # A tibble: 6 x 2
##   title                              n
##   <chr>                          <int>
## 1 Pulp Fiction (1994)            31362
## 2 Forrest Gump (1994)            31079
## 3 Silence of the Lambs, The (1991) 30382
## 4 Jurassic Park (1993)           29360
## 5 Shawshank Redemption, The (1994) 28015
## 6 Braveheart (1995)              26212
```

```
## # A tibble: 6 x 2
##   title                            n
```

```
##   <chr>                            <int>
## 1 Big Fella (1937)                     1
## 2 Condo Painting (2000)                1
## 3 Hellhounds on My Trail (1999)        1
## 4 Quarry, The (1998)                   1
## 5 Stacy's Knights (1982)               1
## 6 Train Ride to Hollywood (1978)       1
```

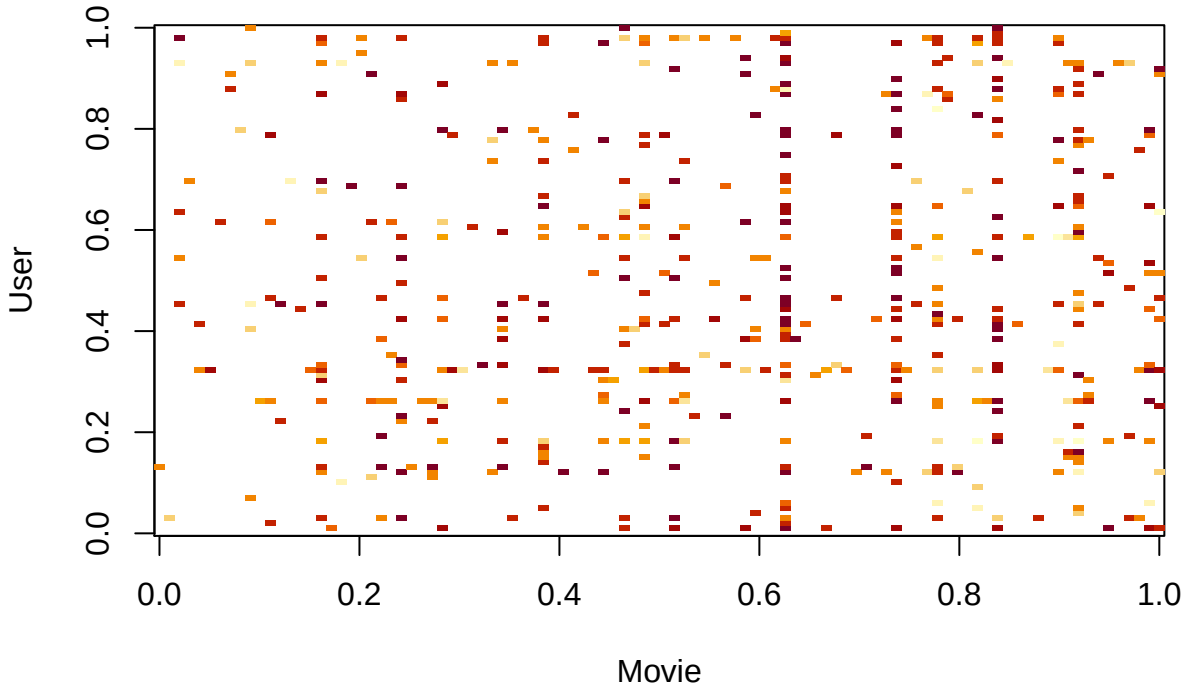
As noted from the histogram, there are more than 100 movies with single ratings. Only 6 of the most obscure movies are shown in the above.

Given the vastly different popularity of the movies, it would be important that we make use of regularization to account for the errors that would be introduced by the movies with very limited ratings.

2.1-5 Sparse matrix

We can reasonably expect that the number of movies seen, and rated, by users would differ. And since the users' taste would be different too we can expect that the matrix of *users* (represented by rows) and *movies* (represented by columns) to be sparse. The number of distinct movies and users in the *edx* set is 10677 and 69878, respectively. This would give a total of 746087406 if each movie has a rating from every user, whereas the total number of ratings in the *edx* set is only 9000055.

```
##   distinct_users distinct_movies total_ratings
## 1          69878           10677       9000055
```



2.2 Models

All but the final model in this project take references from Rafael Irizarry's *Introduction to Data Science* book, supplemented with my own understanding and the codes further built on.

2.2-1 Simple model: Average of ratings

A simple first model would be one that takes the average of ratings for all movies as the prediction:

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

with $\epsilon_{i,u}$ being independent errors centered at 0 and μ being the "true" rating for all movies. Since the estimate that minimizes the RMSE is the least squares estimate of μ, the prediction for μ is the mean of all ratings.

```
#prediction using mean of ratings
mu_hat <- mean(train$rating)
mu_hat
```

```
## [1] 3.5124
```

```
#RMSE of first model
first_rmse <- RMSE(pre_val$rating, mu_hat)
first_rmse
```

```
## [1] 1.0606
```

A tibble of the RMSE is created to keep track of the RMSE results for the different models.

```
#keep track of model results
rmse_results <- tibble(method = "Just the average",
                       RMSE = as.character(round(first_rmse, 5)))
```

2.2-2 Simple model: Added movie effects

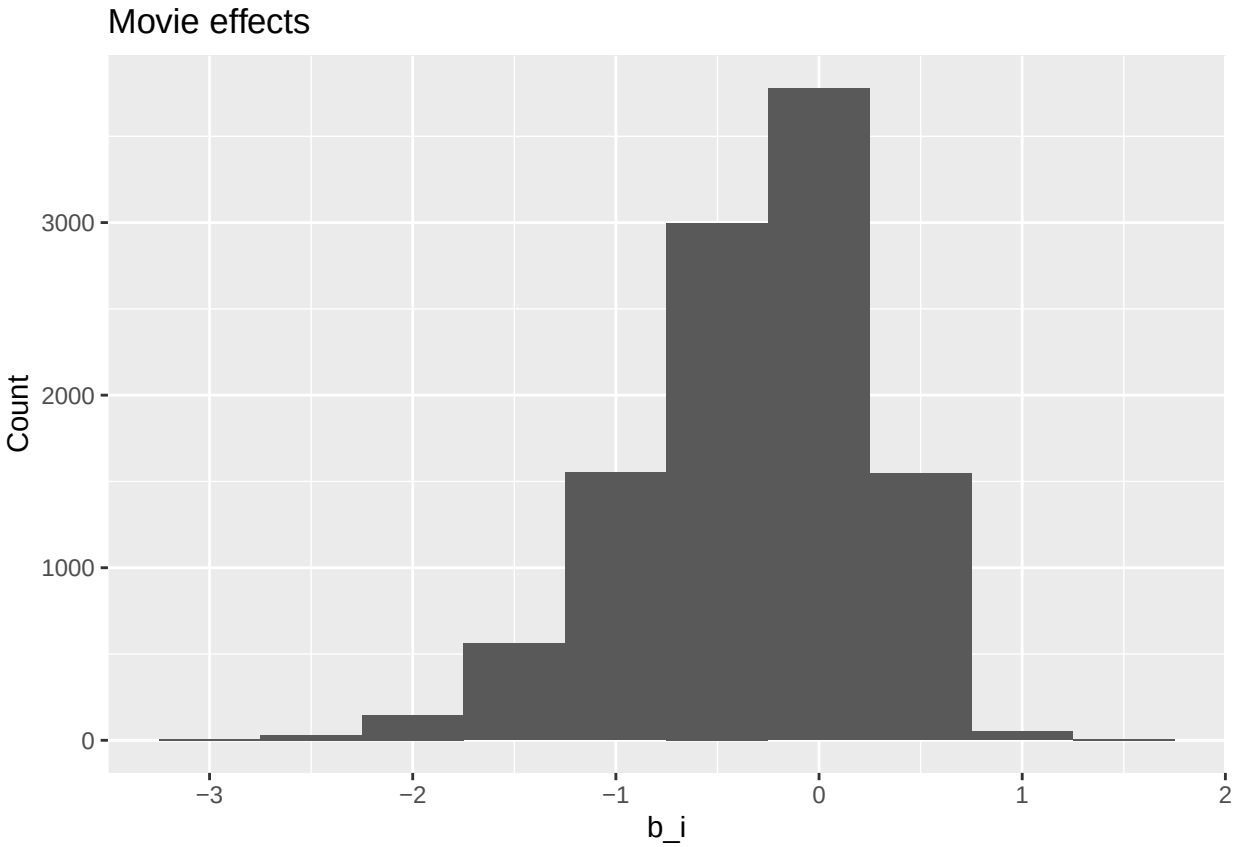
Some movies may tend to have relatively high ratings whereas some movies may have relatively low ratings. So we expect there would be movie effects as they tend to be rated differently. The earlier model equation would now be added a new term:

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

with b_i being the "bias" introduced by the movie effects.

The b_i term can be estimated by the mean of each movie i, i.e. the least square estimate of b_i is the mean of $(Y_{u,i} - \hat{\mu})$ for each movie i.

```
#estimate of movie effects (b_i)
movie_effect <- train %>% group_by(movieId) %>%
                  summarise(b_i = mean(rating - mu_hat))
#plot movie effects (b_i)
movie_effect %>% ggplot(aes(b_i)) + geom_histogram(binwidth = 0.5)  +
                  ggtitle("Movie effects") + xlab("b_i") +ylab("Count")
```



The above shows that most ratings are close to the average of all ratings, but the distribution is left-skewed.

We can now make predictions $\hat{y_{u,i}} = \hat{\mu} + \hat{b_i}$.

```
#make predictions for movies in pre-validation set
pred_b_i <- mu_hat +
              pre_val %>% left_join(movie_effect, by = "movieId") %>% pull(b_i)

#calculate RMSE for the movie effects model
b_i_rmse <- RMSE(pre_val$rating, pred_b_i)
b_i_rmse
```

```
## [1] 0.94381
```

The RMSE for the model with the added movie effects shows a slight improvement compared to the first model.

```
#keep track of model RMSE results
rmse_results <- rmse_results %>% bind_rows(tibble(method = "Added movie effects",
                                                  RMSE = as.character(round(b_i_rmse, 5))))
```

2.2-3 Simple model: Added movie and user effects

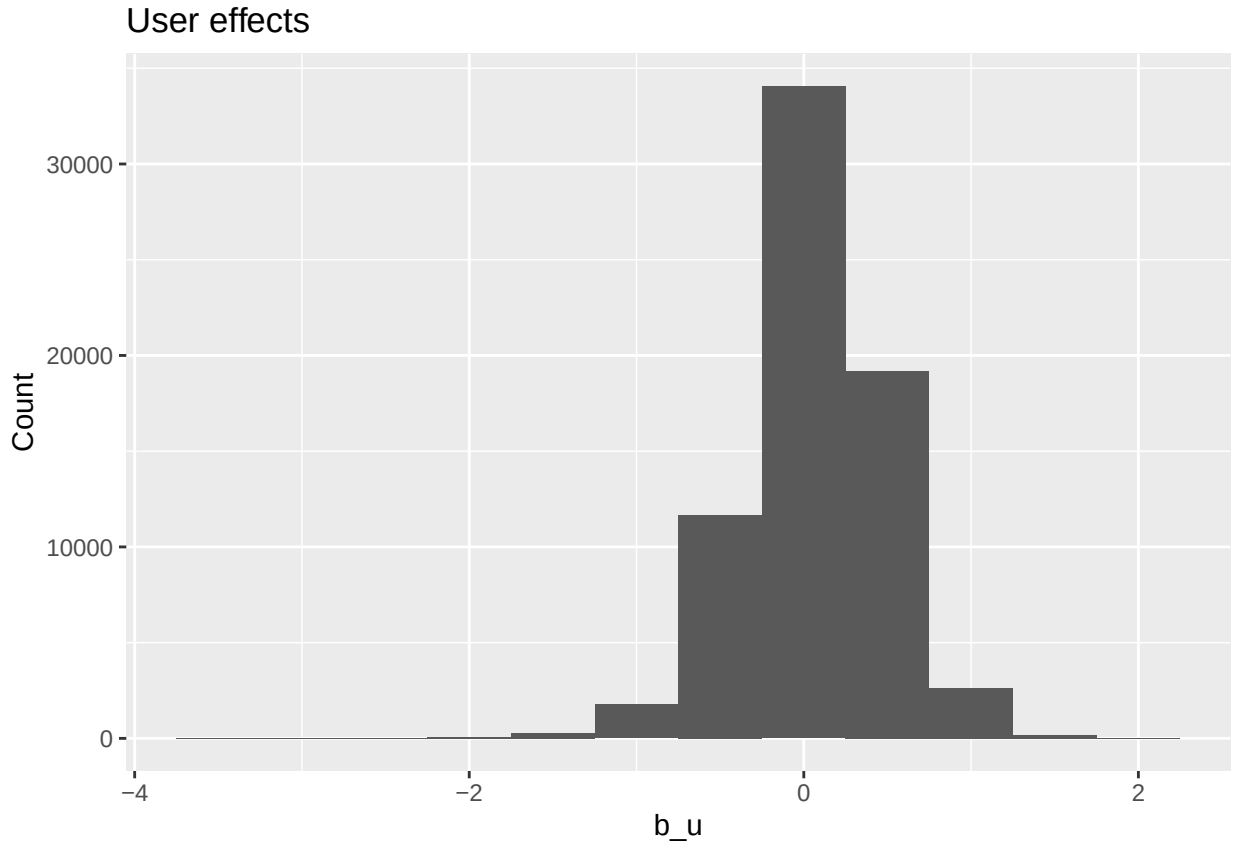
Similarly to the movie effects, we can also reasonably expect that some users may tend to rate movies more critically than the others, and vice-versa. We can introduce another term to the earlier model to account for this effect:

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

with b_u being the "bias" introduced by specific users.

The estimate for b_u is the mean of $(Y_{u,i} - \hat{\mu} - \hat{b_i})$ for each user u.

```
#estimate of user effects (b_u)
user_effect <- train %>% left_join(movie_effect, by = "movieId") %>%
                  group_by(userId) %>%
                  summarise(b_u = mean(rating - mu_hat - b_i))
#plot user effects (b_u)
user_effect %>% ggplot(aes(b_u)) + geom_histogram(binwidth = 0.5)  +
                  ggtitle("User effects") + xlab("b_u") +ylab("Count")
```



The above shows that a majority of the users gave ratings close to the average of all ratings, with very few ratings close to either end of the scale.

```
#make predictions for movies in pre-validation set
pred_b_u <- pre_val %>% left_join(movie_effect, by = "movieId") %>%
              left_join(user_effect, by = "userId") %>%
              mutate(pred = mu_hat + b_i + b_u) %>% pull(pred)

#calculate RMSE for the model with added user effects
b_u_rmse <- RMSE(pre_val$rating, pred_b_u)
b_u_rmse
```

```
## [1] 0.86597
```

The RMSE predicted with both the added movie and user effects shows a further improvement to 0.86597.

```
#keep track of model RMSE results
rmse_results <- rmse_results %>%
                  bind_rows(tibble(method = "Added movie and user effects",
                  RMSE = as.character(round(b_u_rmse, 5))))
```

2.2-4 Regularized model

We know from the earlier analysis that regularization would be needed as the number of ratings for different movies are vastly different between some obscure movies and more popular movies.

With regularization, we would be minimizing the penalized least squares equation:

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

with the first term being the sum of least squares, and the second is penalty that gets larger when b_i's and b_u's are large. The equation for b_i values that minimizes this equation is:

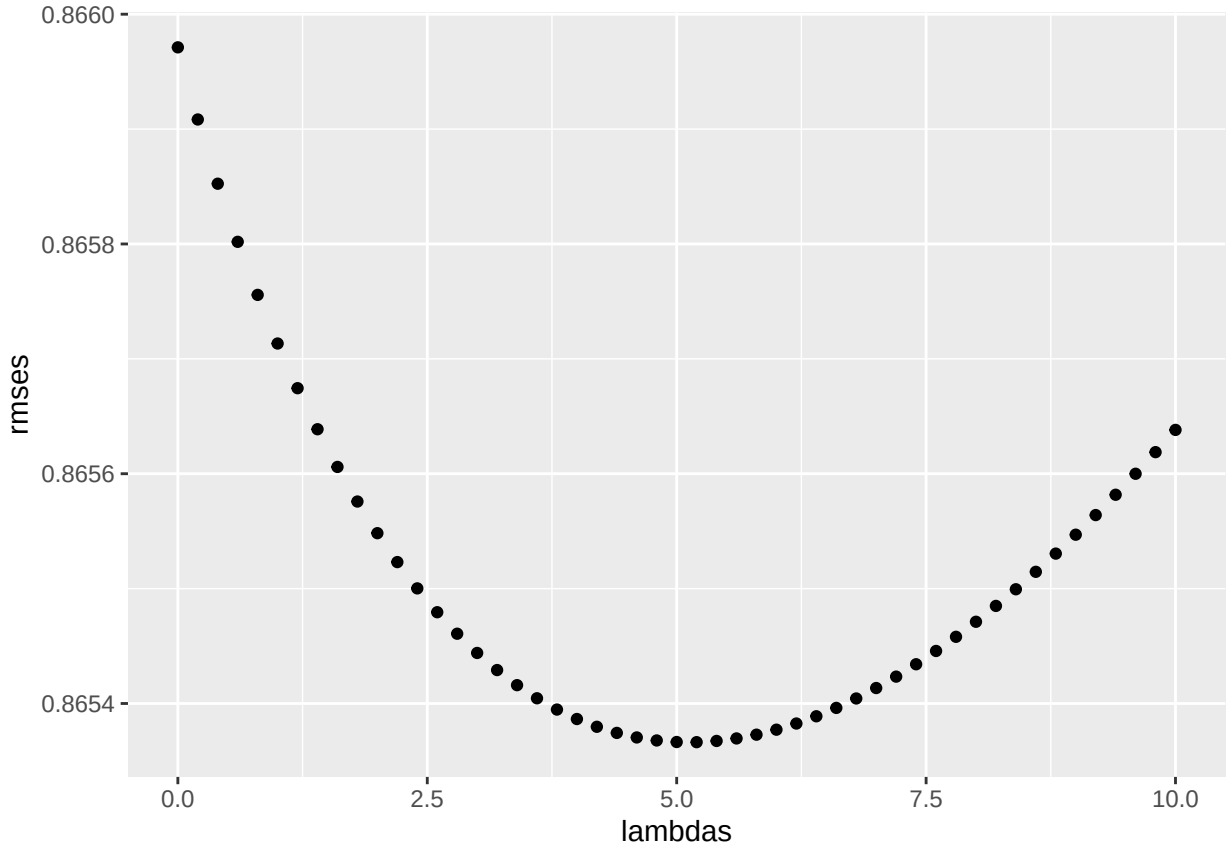
$$\hat{b_i}(\lambda) = \frac{1}{\lambda + n_i}\sum_{u=1}^{n_i}(Y_{u,i} - \hat{\mu})$$

where n_i is the number of ratings made for movie i. The equation for b_u values that minimizes the equation is:

$$\hat{b_u}(\lambda) = \frac{1}{\lambda + n_u}\sum_{i=1}^{n_u}(Y_{u,i} - \hat{\mu} - b_i)$$

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

```
#penalty lambda to be tested
lambdas <- seq(0, 10, 0.2)

#calculate RMSE values with different lambdas
rmses <- sapply(lambdas, function(lambda){
  mu <- mean(train$rating)

  b_i <- train %>% group_by(movieId) %>%
            summarise(b_i = sum(rating - mu)/(n()+lambda))

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

  predictions <- pre_val %>% left_join(b_i, by = "movieId") %>%
                  left_join(b_u, by = "userId") %>%
                  mutate(pred = mu + b_i + b_u) %>% pull(pred)

  RMSE(predictions, pre_val$rating)
})

#plot RMSE values with lambdas
qplot(lambdas, rmses)
```



The optimal penalty lambda value and the corresponding RMSE:

```
#optimal lambda value
lambda <- lambdas[which.min(rmses)]
#corresponding RMSE
regularized_rmse <- min(rmses)

tibble("Optimal lambda" = lambda, "Corresponding RMSE" = regularized_rmse)
```

```
## # A tibble: 1 x 2
##   `Optimal lambda` `Corresponding RMSE`
##              <dbl>                <dbl>
## 1              5.2                0.865
```

The regularized model's RMSE is 0.86537. This RMSE is close to (but still higher than) the target RMSE of less than 0.86490. Note that the RMSE obtained is not tested against the final hold-out set (the *validation* set). If we do test the predictions by the regularized model against the final hold-out set, it would likely fall short of the target RMSE.

A final model is attempted next before testing our selected model against the *validation* set.

```
#keep track of model RMSE results
rmse_results <- rmse_results %>%
                  bind_rows(tibble(method = "Regularized with movie and user effects",
                  RMSE = as.character(round(regularized_rmse, 5))))
```

2.2-5 Matrix factorization

The regularized model with user and movie effects accounts for the effects of the different number of ratings for movies as well as the effects of the different number of ratings by users. However, we have not yet taken into consideration that groups of similar movies (in certain aspects) would have similar patterns to them, and groups of similar users would also have similar patterns to how they would rate movies. Matrix factorization would help to address this.

Matrix factorization is related to factor analysis, singular value decomposition (SVD), and principal component analysis (PCA) Introduction to Data Science. Given a sparse matrix, matrix factorization works by approximating the whole matrix $R_{m\times n}$ by the product of two matrices of lower dimensions, $P_{k\times m}$ and $Q_{k\times n}$, such that

$$R \approx P^T Q$$

.

In the context of the recommendation system, we can write

$$r_{u,i} \approx p_u q_i$$

where $r_{u,i}$ is the residual in the following equation:

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

In more general terms, the residuals may be written to indicate the matrix dimensions:

$$R_{n_u\times n_i} \approx (P^T)_{n_u\times d} Q_{d\times n_i}$$

where d being the number of latent factors, e.g. genres of movies or other aspects like movies starring a certain actor/actress.

I will be using the *recosystem* package, which is an R wrapper of the open source library *LIBMF* for recommender system using parallel factorization. The following code is computationally expensive, and a more optimal set of parameters can be determined based on consideration of a balance between computational cost and required accuracy. But this is not done in the current report.

```
set.seed(11, sample.kind="Rounding")

#create a model object
r = Reco()

#specify data sets from R objects (the other option being specify from file)
train_reco <- data_memory(user_index = train$userId, item_index = train$movieId,
                          rating = train$rating)
pre_val_reco <- data_memory(user_index = pre_val$userId, item_index = pre_val$movieId,
                            rating = pre_val$rating)

#tuning parameters (edit nthread argument based on your hardware)
#keeping default values for both L1 and L2 for user and item factors
opts = r$tune(train_reco, opts = list(dim = seq(20, 30, 5), lrate = c(0.1, 0.15),
                                      nthread = 14, niter = 20))

#train model
r$train(train_reco, opts = c(opts$min, nthread = 14, niter = 20))
```

```
## iter      tr_rmse          obj
```

```
##    0       0.9036   9.8352e+06
##    1       0.8887   9.1890e+06
##    2       0.8448   8.3869e+06
##    3       0.8213   7.9818e+06
##    4       0.8039   7.7027e+06
##    5       0.7908   7.5025e+06
##    6       0.7805   7.3576e+06
##    7       0.7719   7.2410e+06
##    8       0.7640   7.1386e+06
##    9       0.7576   7.0590e+06
##   10       0.7518   6.9890e+06
##   11       0.7464   6.9277e+06
##   12       0.7424   6.8851e+06
##   13       0.7383   6.8377e+06
##   14       0.7344   6.7974e+06
##   15       0.7308   6.7614e+06
##   16       0.7276   6.7267e+06
##   17       0.7251   6.7030e+06
##   18       0.7226   6.6790e+06
##   19       0.7204   6.6592e+06
```

```
#predictions for pre-validation set
pred_reco <- r$predict(pre_val_reco, out_memory())

#RMSE against pre-validation set
rmse_reco <- RMSE(pre_val$rating, pred_reco)
rmse_reco
```

```
## [1] 0.78717
```

The RMSE value against the pre-validation set of 0.78717 is a substantial improvement over my previous models. The *recosystem* model based on matrix factorization is our selected model. We will update the RMSE results before moving on.

```
#keep track of model RMSE results
rmse_results <- rmse_results %>%
                  bind_rows(tibble(method = "Matrix factorization (recosystem)",
                  RMSE = as.character(round(rmse_reco, 5))))
```

3 RESULTS

3.1 Model results

The first model using just the average of all movie ratings has an RMSE of 1.06059. The RMSE can be interpreted similarly to a standard deviation, i.e. it is the typical error we expect to get when making a prediction. For an RMSE of more than 1, it means that we would typically get an error that is larger than one star.

The subsequent models added movie as well as user effects, since there would be movies that are generally rated higher and vice-versa, and similarly there would be users who are generally generous with ratings and vice-versa. The model with added movie effects only gets an improvement with an RMSE of 0.94381, and

the RMSE improves to 0.86597 once we have also included the user effects. The results in an approximately 0.2 star improvement from the first model.

Added to the fact that there are obscure movies which have very few ratings and users who gave very few ratings, regularization is required to account for the unwanted effects:

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

With the penalty terms in the least squares equation to be minimized, the estimates for b_i and b_u get shrunken towards zero when the sample sizes n_i and n_u are small, whereas very large samples sizes give us stable estimates and the penalty λ effectively has no effects. In the case for this project, regularization has only resulted in a slight improvement in RMSE to 0.86537.

The final model takes into account of the fact that there would be groups of users and movies that exhibit certain patterns in how they rate, or how the movies are rated. This makes use of matrix factorization and the *recosystem* package is used for this model. This results in a substantial improvement in RMSE to 0.78717.

```
#summary of RMSE results for all models tested against "pre-validation" set
rmse_results
```

```
## # A tibble: 5 x 2
##   method                                RMSE
##   <chr>                                 <chr>
## 1 Just the average                      1.06059
## 2 Added movie effects                   0.94381
## 3 Added movie and user effects          0.86597
## 4 Regularized with movie and user effects 0.86537
## 5 Matrix factorization (recosystem)     0.78717
```

All the above RMSE results are evaluated against the "pre-validation" set, which is sampled randomly from the *edx* set prescribed in the project assignment.

3.2 Final hold-out set validation

As we have our selected recommender system in the Matrix Factorization (recosystem) model, we can now test its performance against the final hold-out set (the *validation* set). Based on the RMSE value tested against the pre-validation set, the model should easily do better than the target RMSE of 0.86490.

The RMSE result tested against the *validation* set is very close to the RMSE value calculated based on the "pre-validation" set, which means that there is no indication of over- or under-training.

```
#specify data sets from R objects (the other option being specify from file)
validation_reco <- data_memory(user_index = validation$userId,
                               item_index = validation$movieId,
                              rating = validation$rating)

#predictions using selected model (recosystem)
pred_final <- r$predict(validation_reco, out_memory())

#RMSE against validation set (final hold-out set)
rmse_final <- RMSE(validation$rating, pred_final)
rmse_final
```

```
## [1] 0.78673
```

The above shows an RMSE value of 0.78673, which is indeed lower than our target RMSE of 0.86490.

4 CONCLUSION

4.1 Summary

A total of five models were trained and evaluated, starting from a simple model that makes use of the average of all movie ratings, incrementally improved upon with movie and user effects as well as regularization, to the final selected model based on matrix factorization. The final selected model was chosen based on the RMSE values for the different models evaluated against a "pre-validation" set, which is part of the *edx* set prescribed in this project assignment to be used for model training and model selection. The prescribed *validation* set is the final hold-out set used for model performance evaluation.

The final RMSE result for the selected model of 0.78673 is substantially lower than our target RMSE of 0.86490.

```
#Summary table of models and corresponding RMSE tested against "pre-validation" set
rmse_results
```

```
## # A tibble: 5 x 2
##   method                                RMSE
##   <chr>                                 <chr>
## 1 Just the average                      1.06059
## 2 Added movie effects                   0.94381
## 3 Added movie and user effects          0.86597
## 4 Regularized with movie and user effects 0.86537
## 5 Matrix factorization (recosystem)     0.78717
```

```
#RMSE of final selected model tested against "validation" set
rmse_final
```

```
## [1] 0.78673
```

4.2 Limitations

The final selected model based on matrix factorization is relatively computationally expensive compared to the other simpler models. The current data sets based on a reduced subset of the database should run fairly well on modern desktop computers or laptops. However, this could present a problem for very large data sets.

In addition, the matrix factorization model should be run every time a new user or movie rating is added to the data sets. Since the model is quite computationally expensive, this may pose an operational issue as to how frequently the model needs to be refreshed. Optionally, (though this would not really address the costly computations issue) additional tuning could be done before operationalization to arrive at a more optimal choice of parameters to be used in training and prediction.

4.3 Future work

The selected model is implemented using the *recosystem* package, which is an easy-to-use R wrapper of the open source library *LIBMF* for recommender system using parallel factorization. However, the performance in terms of RMSE should reasonably be further improved by applying different recommender algorithms. This would be possible with the R extension package *recommenderlab* provides a general research infrastructure to test and develop recommender algorithms including UBCF, IBCF, FunkSVD and association rule-based algorithms.