Movie Recommendation Project

Andrew Yu

2022-06-03

Movie Recommendation Project

By Andrew Yu

Libraries and Settings

Introduction

Movie recommendation models have become a big part of data science in media companies recently. With streaming platforms such as Netflix, HBO, Hulu, Amazon Video, etc., companies have sought the best models to recommend movies/shows to their viewers to increase viewership, subscription retention, and more. To model after this real-life application, I utilize several machine learning models to create the most accurate predictions/recommendations.

Data

The data used to create this model is the MovieLens dataset provided by GroupLens Research. It was collected over multiple time periods. We utilize their 10 million movie ratings dataset which includes 10,000 different movies and 72,000 users which was released in January 2009. The dataset is gathered as such.

Methods/Analysis

We first utilize a basic model that assumes/predicts the average rating to be the rating for all movies. We then build upon this model factoring in effects of different features and taking into account how they would adjust the baseline rating (average rating). We also utilize a Matrix Factorization model to see if this is better.

- **Model 1:** Naive Model
- **Model 2:** Genre Effect Model
- **Model 3:** Year of Release Effect Model
- **Model 4:** Movie Effect Model
- **Model 5:** User-Movie Effect Model
- **Model 6:** Regularized Movie Effect Model
- **Model 7:** Regularized User-Movie Effect Model
- **Model 8:** Matrix Factorization Model

Data Pre-Processing

First we want to extract the desired variables and drop unneeded ones from the dataset. Then, we also want to split up our EdX set into test and training so we can develop the best model to implement. I select 2 final models to use as a recommendation system which I validate by the validation dataset. We seek models that minimizes the Root Mean Squared Error (RMSE) as much as possible.

Models

Model 1: Naive Model This model is the most basic model and merely serves as a model to build off of. Here we find the average rating of our training set and predict all of the ratings to be the average and all differences explained by random variation. This model also assumes that the errors are independent and are sampled from the same distribution, centered at 0. The model looks as such.

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

```
mu <- mean(train$rating)
naive_rmse <- RMSE(test$rating, mu)
results <- tibble(Model = "Naive", RMSE = naive_rmse)
results
```

```
## # A tibble: 1 x 2
##   Model  RMSE
##   <chr> <dbl>
## 1 Naive  1.06
```

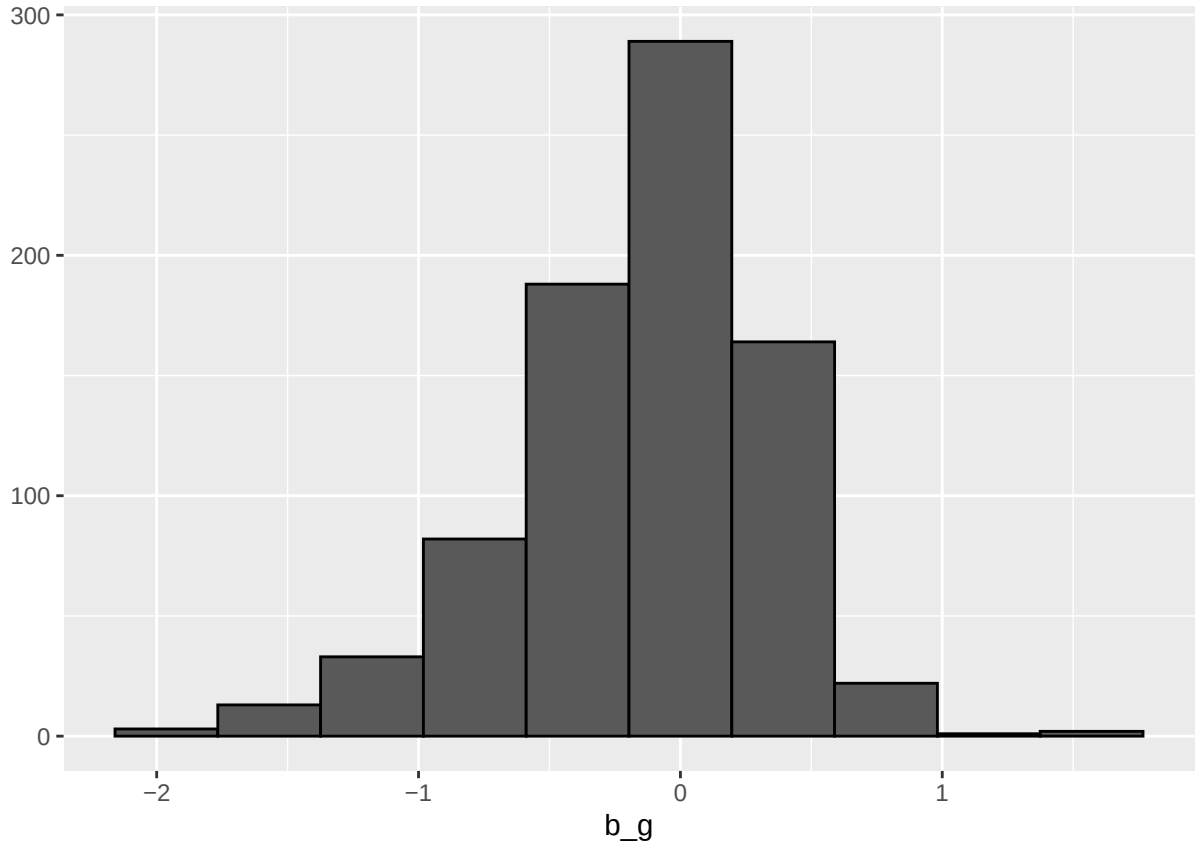
As we can see, the RMSE is quite high for this model at 1.06, which is to be somewhat expected.

Model 2: Genre Effects Model Since we also know that people have a tendency towards different types of genres, we try to implement a genre effect onto this model. The best way to capture this would be to encode the genre column because it is a multi-label classification variable. This means a movie can identify with 1 or more genre variables. However, due to the lack of technical capabilities of the computer's strength, this was unable to be done as there were 24 unique genres. Instead, we try to make do with the basic multi-levels of genre. The model is as following.

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

```
genre_avgs <- train %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu))

qplot(b_g, data = genre_avgs, bins = 10, color = I("black"))
```




```
predicted_ratings <- mu + test %>%
  left_join(genre_avgs, by='genres') %>%
  pull(b_g)

genre_rmse <- RMSE(predicted_ratings, test$rating)

results <- bind_rows(results,
                     tibble(Model="Genre Effects Model",
                            RMSE = genre_rmse))
rm(genre_rmse)
results
```

```
## # A tibble: 2 x 2
##   Model                RMSE
##   <chr>               <dbl>
## 1 Naive                1.06
## 2 Genre Effects Model  1.02
```

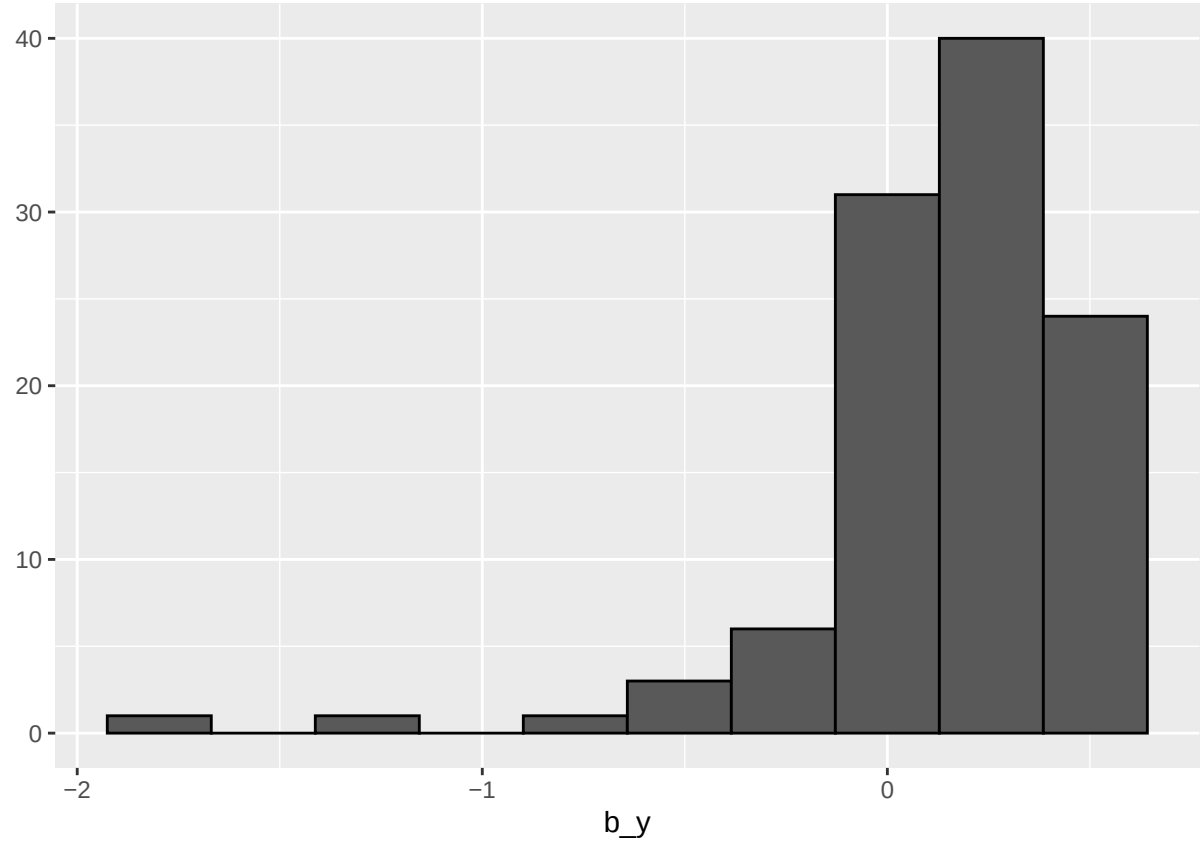
The RMSE does improve to 1.018, however, this process is a bit flawed and would not be reusable in a mass scale level due to computing time and accuracy.

Model 3: Year of Release Effect Model We also test for the effect of the movie's year of release. Are there certain years or a time period in which there were "better" movies? This is what we are trying to capture with this effect implemented into the model. The model is as following.

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

```
release_avgs <- train %>%
  group_by(release) %>%
  summarize(b_y = mean(rating - mu))

qplot(b_y, data = release_avgs, bins = 10, color = I("black"))
```




```
predicted_ratings <- mu + test %>%
  left_join(release_avgs, by='release') %>%
  pull(b_y)

release_rmse <- RMSE(predicted_ratings, test$rating)
results <- bind_rows(results,
                     tibble(Model="Release Year Effects Model",
                            RMSE = release_rmse))
rm(release_rmse)
results
```

```
## # A tibble: 3 x 2
##   Model                        RMSE
##   <chr>                       <dbl>
## 1 Naive                        1.06
## 2 Genre Effects Model          1.02
## 3 Release Year Effects Model   1.05
```

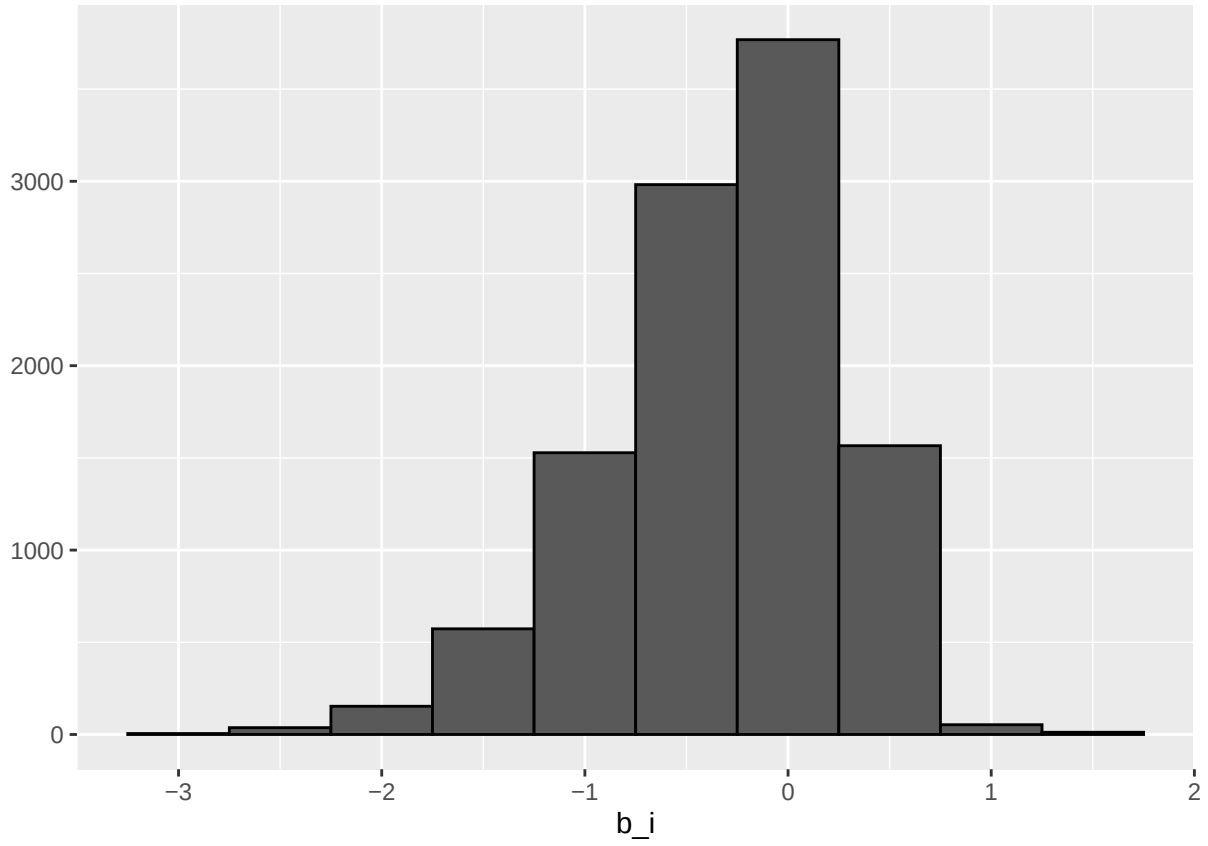
As we see, it does have some effect but its effect is even lower than the faulty genre effect model we calculated.

Model 4: Movie Effects Model Some movies are generally considered "better" than others. To take this into account in our model, we take into account movie effects in this model to see if this helps our predictive power. The model is as following.

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

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

qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```




```
predicted_ratings <- mu + test %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

movie_rmse <- RMSE(predicted_ratings, test$rating)
results <- bind_rows(results,
                     tibble(Model="Movie Effects Model",
                                RMSE = movie_rmse))
rm(movie_rmse)
results
```

```
## # A tibble: 4 x 2
##   Model                         RMSE
##   <chr>                        <dbl>
## 1 Naive                        1.06
## 2 Genre Effects Model          1.02
## 3 Release Year Effects Model 1.05
## 4 Movie Effects Model          0.944
```

So far, the movie effects seems to have the best ability in helping improve our predictive powers, having a RMSE of 0.9437.

Model 5: User-Movie Effects Model Now, even if the movie is a good or great movie, everyone has a different opinion and there was some context as to when or how they decided to rate that movie. We want to take into account this user specific effect on movies whilst still including the movie effect.

Let's check and see where our movie effects model had the 10 biggest errors/residuals.

```
test %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  slice(1:10) %>%
  pull(title)
```

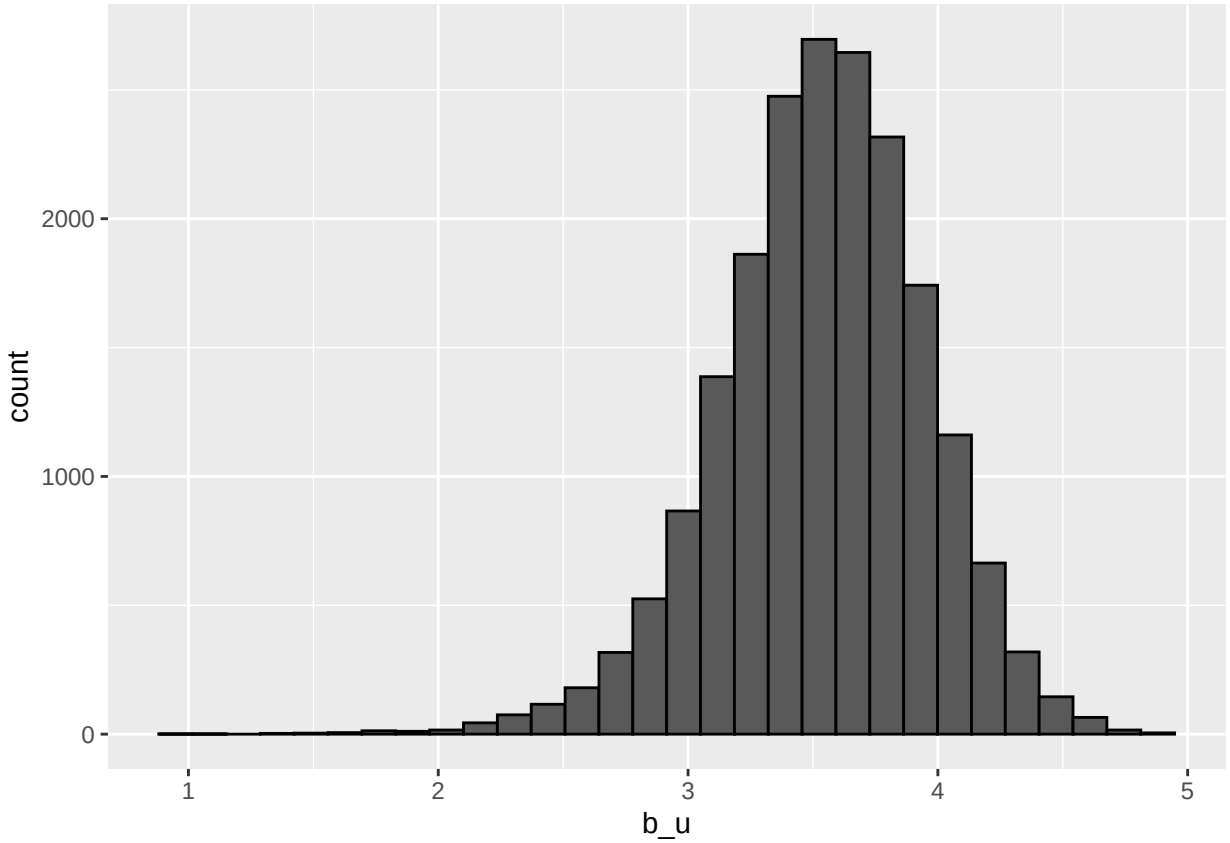
```
##  [1] "From Justin to Kelly (2003)"      "Time Changer (2002)"
##  [3] "Shawshank Redemption, The (1994)" "Shawshank Redemption, The (1994)"
##  [5] "Shawshank Redemption, The (1994)" "Shawshank Redemption, The (1994)"
##  [7] "Shawshank Redemption, The (1994)" "Shawshank Redemption, The (1994)"
##  [9] "Shawshank Redemption, The (1994)" "Children Underground (2000)"
```

Interestingly enough it shows that although some movies are obscure, one movie seems to have been rated very differently user to user, which further supports our reasoning to include a user-specific effect. As we can see here for the average rating for users who have rated 100 or more movies, there is substantial variability in how they rated a movie on average. Thus, it can be implied that it is worth modeling into our movie effects model, a user specific effect. The model is as following.

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

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

train %>%
  group_by(userId) %>%
  filter(n()>=100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```




```
predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

userMovie_rmse <- RMSE(predicted_ratings, test$rating)
results <- bind_rows(results,
                     tibble(Model="User-Movie Effects Model",
                            RMSE = userMovie_rmse))
rm(userMovie_rmse)
results
```

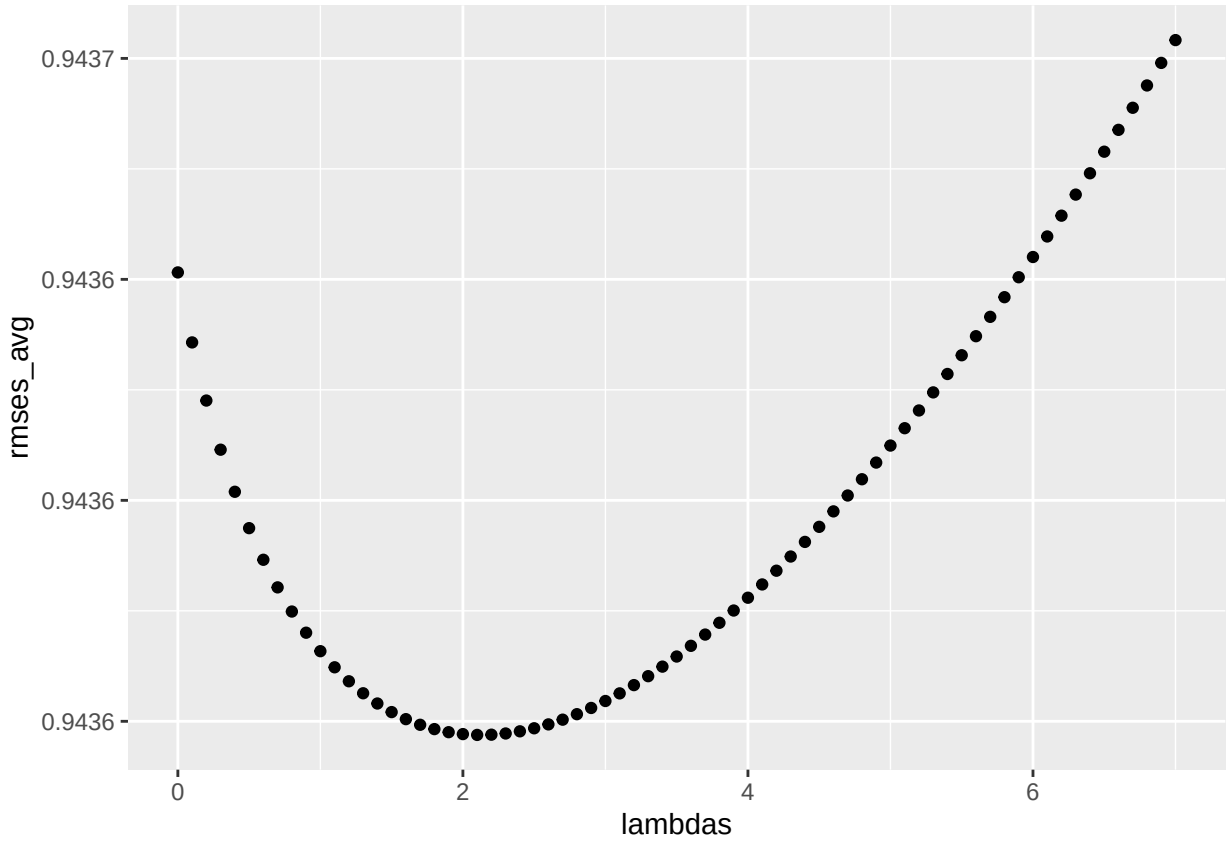
```
## # A tibble: 5 x 2
##   Model                        RMSE
##   <chr>                       <dbl>
## 1 Naive                       1.06
## 2 Genre Effects Model         1.02
## 3 Release Year Effects Model 1.05
## 4 Movie Effects Model         0.944
## 5 User-Movie Effects Model    0.866
```

After accounting for User-Movie Specific effects, we obtain a much better model with a smaller RMSE, the smallest thus far.

Model 6: Regularized Movie Effect Model For the models we have just built, we want to then further improve these by penalizing the effects by either the movie effect or user-specific effect when they are large. By doing so, we constrain the variability. The model then looks to minimize.

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

To do so, we utilize a K-Fold Cross Validation to go ahead and find the best lambda to regularize the movie effects model.



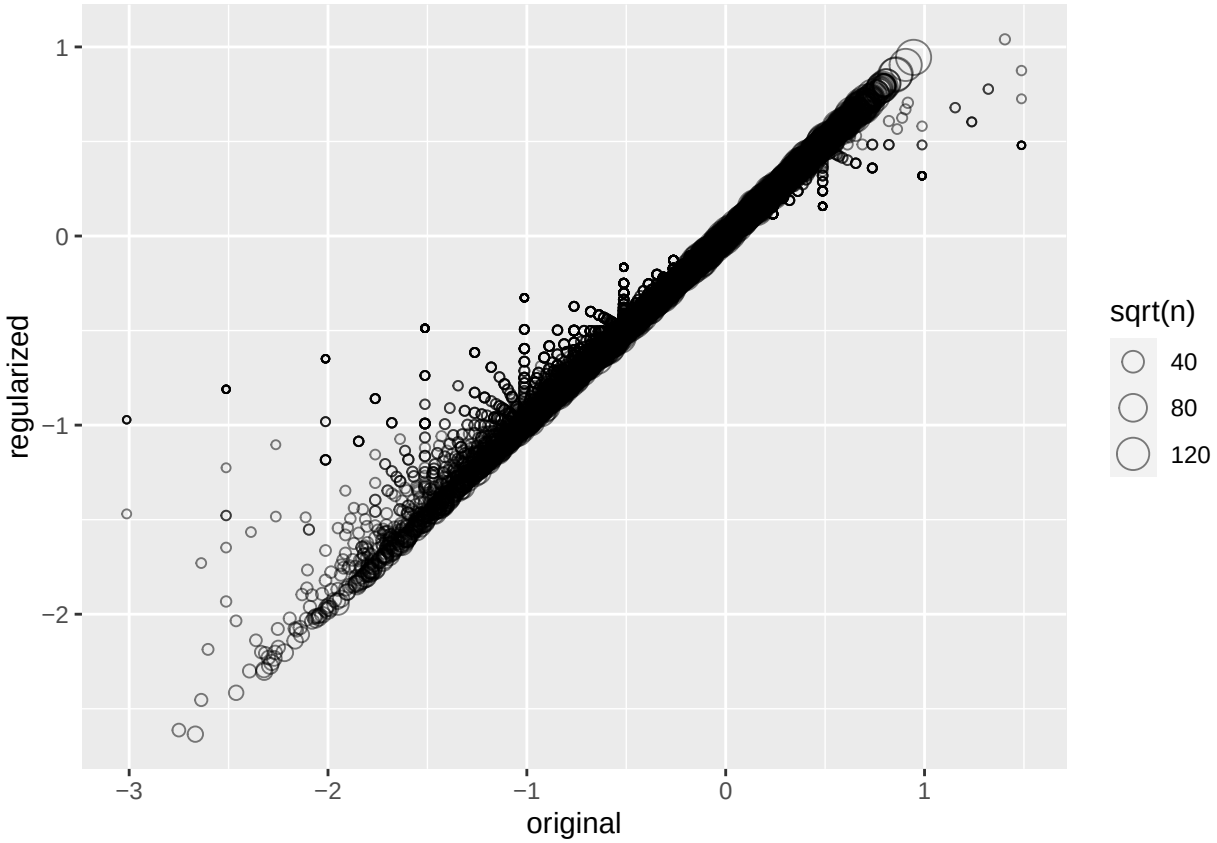

```
lambda <- lambdas[which.min(rmses_avg)]
lambda
```

```
## [1] 2.1
```

We see that 2.1 is the optimal value of lambda. Using that value, we go ahead and utilize the following movie effects model.

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

tibble(original = movie_avgs$b_i,
       regularized = movie_reg_avgs$b_i,
       n = movie_reg_avgs$n_i) %>%
  ggplot(aes(original, regularized, size=sqrt(n))) +
  geom_point(shape=1, alpha=0.5)
```



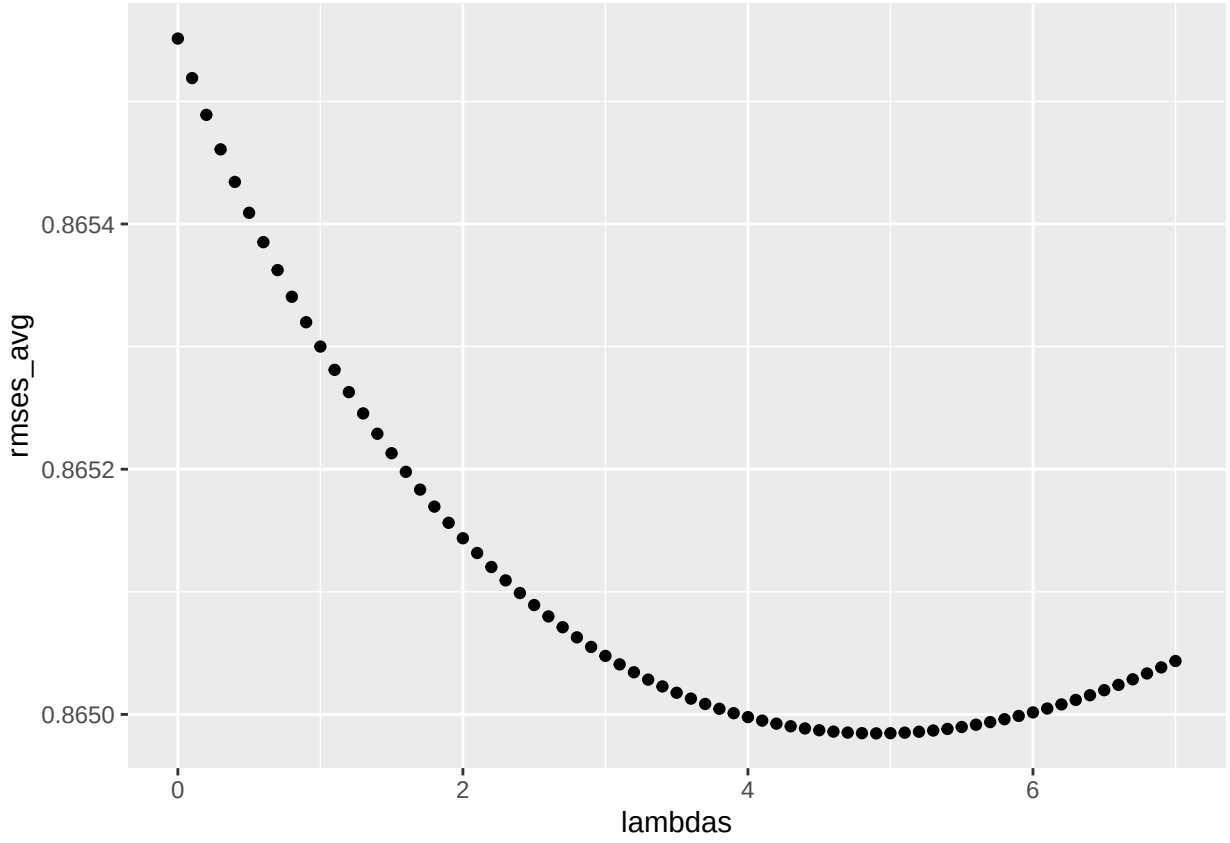
We can see how by regularizing, the estimates have shrunk.

```
predicted_ratings <- test %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
regMovie_rmse <- RMSE(predicted_ratings, test$rating)
results <- bind_rows(results,
                     tibble(Model="Regularized Movie Effects Model",
                            RMSE = regMovie_rmse))
rm(regMovie_rmse)
results
```

```
## # A tibble: 6 x 2
##   Model                            RMSE
##   <chr>                           <dbl>
## 1 Naive                           1.06
## 2 Genre Effects Model             1.02
## 3 Release Year Effects Model      1.05
## 4 Movie Effects Model             0.944
## 5 User-Movie Effects Model        0.866
## 6 Regularized Movie Effects Model 0.944
```

However, we see that it does not effect it to an extent where the RMSE really improves.

Model 7: Regularized User-Movie Effects Model We can utilize this technique of regularizing the user-movie effects model too and see if that does a better job.




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

```
## [1] 4.9
```

We see that through a 10-fold cross validation, we obtain an optimal lambda of 4.9 for the regularized user-movie effects model.

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

b_u <- train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda), n_i = n())
predicted_ratings <- test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
regUserMovie_rmse2 <- RMSE(predicted_ratings, test$rating)
```

```
results <- bind_rows(results,
                     tibble(Model="Regularized User-Movie Effects Model",
                            RMSE = regUserMovie_rmse2))
results
```

```
## # A tibble: 7 x 2
##   Model                                 RMSE
##   <chr>                                <dbl>
## 1 Naive                                1.06
## 2 Genre Effects Model                  1.02
## 3 Release Year Effects Model           1.05
## 4 Movie Effects Model                  0.944
## 5 User-Movie Effects Model             0.866
## 6 Regularized Movie Effects Model      0.944
## 7 Regularized User-Movie Effects Model 0.865
```

We can see that even despite regularizing, the model still does not improve significantly.

Model 8: Matrix Factorization Model We obtained a substantial improvement in our effects models through a regularized user-movie effects model. However, we want to see if another algorithm can be more accurate in our predictions. Here we utilize Matrix Factorization, a collaborative filtering algorithm. To implement matrix factorization we utilize the 'recosystem' package that allows for us to conduct matrix factorization utilizing much less memory usage.

First, we split our data into a sparse matrix and formulate a train and test dataset. Then we utilize 9 different combinations of factors and iterations to find the most optimal tuning parameters for our matrix factorization.

```
train_data <- data_memory(user_index = train$userId,
                          item_index = train$movieId,
                          rating = train$rating, index1 = TRUE)
test_data <- data_memory(user_index = test$userId,
                          item_index = test$movieId,
                          rating = test$rating, index1 = TRUE)
recommender <- Reco()

factors <- seq(10,30,10)
iterations <- seq(100,500,200)
args <- expand.grid(x = factors, y = iterations)

mf_tuning <- mapply(function(x,y){
  recommender$train(train_data, opts = c(dim = x, costp_l2 = 0.1, costq_l2 = 0.1,
                                         lrate = 0.1, niter = y, nfold = 10,
                                         nthread = 6, verbose = F))
  test$prediction <- recommender$predict(test_data, out_memory())
  return(c(f = x, i = y, mf_rmses = RMSE(test$prediction, test$rating)))
}, x = args$x, y = args$y)
mf_tuning <- as.data.frame(t(mf_tuning))
mf_tuning[which.min(mf_tuning$mf_rmses),]
```

```
##    f   i mf_rmses
## 9 30 500   0.8099
```

We see that 30 factors with 500 iterations seems to produce the lowest RMSE.

Results

To now validate our model and see the results, I chose the regularized user-movie effects model (in case some people's hardware may not be able to run the other one) and the matrix factorization model.

We validate our models by utilizing the models with the parameters we chose on the EdX and Validation data sets.

```
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+4.9))
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+4.9), n_i = n())
predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model1 <- RMSE(predicted_ratings, validation$rating)
final_results <- tibble(Model= 'Regularized User-Movie Effects Model',
                        RMSE = model1)
final_results
```

Model 1: Regularized User-Movie Effects Model

```
## # A tibble: 1 x 2
##   Model                                RMSE
##   <chr>                               <dbl>
## 1 Regularized User-Movie Effects Model 0.865
```

```
edx_data <- data_memory(user_index = edx$userId,
                        item_index = edx$movieId,
                        rating = edx$rating, index1 = TRUE)
validation_data <- data_memory(user_index = validation$userId,
                       item_index = validation$movieId,
                       rating = validation$rating, index1 = TRUE)
recommender <- Reco()
recommender$train(edx_data, opts = c(dim = 30, costp_l2 = 0.1, costq_l2 = 0.1,
                                     lrate = 0.1, niter = 500, nfold = 10,
                                     nthread = 6, verbose = F))
validation$prediction <- recommender$predict(validation_data, out_memory())
model2 <- RMSE(validation$prediction, validation$rating)
final_results <- bind_rows(final_results,
                           tibble(Model= 'Matrix Factorization Model',
                                  RMSE = model2))
final_results
```

Model 2: Matrix Factorization Model

```
## # A tibble: 2 x 2
##   Model                                RMSE
##   <chr>                               <dbl>
## 1 Regularized User-Movie Effects Model 0.865
## 2 Matrix Factorization Model           0.809
```

Conclusion

In conclusion, the matrix factorization model does provide the best predictive powers and the lowest RMSE. However, some issues with it is the computing powers necessary to test it and also tune its parameters. There are more parameters to tune using the 'recosystem' package as well such as learning rate and lambdas. The baseline regularized user-movie effects model doesn't do as well as the MF model, but is much quicker to utilize and easier to test out its parameters. Although the cross validation takes a while, that is consistent amongst both models. The biggest limitations to this project was the computing powers of my system, but I feel with a stronger system, these models could be much improved and perhaps other models also utilized to see if they possibly create better predictions as well.