# Movielens Recommendation Algorithm

Alix Benoit

6/10/2020

## Introduction

This project makes use of the movielens 10 million dataset from grouplens at the University of Minnesota. This dataset is comprised of 10000054 total ratings of 10681 movies by 71567 users acquired from the online movie recommendation service "Movielens". Users were selected at random, and all of them had rated at least 20 movies.

The following libraries were used:
- tidyverse
- caret
- knitr
- recosystem

The following demographics are included for each rating in the dataset:
- The ID of the user
- The ID of the movie
- The rating (number of stars given: 1 = doesn't like the movie; 5 = loves it)
- When the review was written (given as a numerical timestamp)
- The title of the movie
- The possible genres the movie could fall into

While there exist larger datasets (such as the 100 million set) the 10 million set was chosen in order to make calculations feasable on an old laptop computer.

The goal of the project is to create a machine learning algorithm that can accurately predict the rating a user will give a certain movie when provided with the demographics above (minus rating of course).

The Root Mean Squared Error (RMSE) was chosen as the measure of accuracy for the algorithm, and the dataset was split into two dataframes:
- The edx dataframe used for training and cross validation of the algorithm, which makes up 90% of the total dataset
- The validation dataframe used only for the final testing of the final model. The target RMSE was 0.86490.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{Y_i} - Y_i)^2}$$

```
rmse_target <- .89490
```

The final model essentially consited of the mean rating plus regularized movie and user effects, aswell as matrix factorization using a parallel Stochastic Gradient Descent.

Here is a sample of the edx data frame showing the first 6 entries:

```r
knitr::kable(head(edx))
```

| | userId | movieId | rating | timestamp | title | genres |
|---|---|---|---|---|---|---|
| 1 | 1 | 122 | 5 | 838985046 | Boomerang (1992) | Comedy\|Romance |
| 2 | 1 | 185 | 5 | 838983525 | Net, The (1995) | Action\|Crime\|Thriller |
| 4 | 1 | 292 | 5 | 838983421 | Outbreak (1995) | Action\|Drama\|Sci-Fi\|Thriller |
| 5 | 1 | 316 | 5 | 838983392 | Stargate (1994) | Action\|Adventure\|Sci-Fi |
| 6 | 1 | 329 | 5 | 838983392 | Star Trek: Generations (1994) | Action\|Adventure\|Drama\|Sci-Fi |
| 7 | 1 | 355 | 5 | 838984474 | Flintstones, The (1994) | Children\|Comedy\|Fantasy |

## Method and Analysis

### Data wrangling and cleaning

Minimal amounts of data cleaning and wrangling are required in order to start data exploration, as a starting script is already given, which creates the Edx and validation sets.

The first step is to split the edx set further, into a "train" set and a "test" set:

```r
i <- createDataPartition(edx$rating, times = 1, p = .1, list = F)
train <- edx[-i,]
test <- edx[i,]
```

And to make sure all the movies and users in the test set are also in the train set:

```r
test <- test %>% semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")
```

The train set is used to train all algorithms and perform cross validation; the test* set is used to get an estimate of how well a particular algorithm performs, and for cross validation purposes as well.

*Note that this is different than the validation set, which is used only on the final model predictions, and cannot be used to make any decisions. Since the test set is derived from the edx set, it can be used to make decisions.

### Baseline model

The following function is written in order to calculate the RMSE from a set of predictions, and a set of actual ratings:

```r
RMSE <- function(actual_rating, predicted_rating){
  sqrt(mean((actual_rating - predicted_rating)^2))
}
```

As a baseline for other models, a simple "guessing" model is adopted, using the mean rating from the train set.

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

This model achieves the following RMSE on the test set:

```r
mu <- mean(train$rating)
rmse_guess <- RMSE(test$rating, mu)
rmse_guess
```

```
## [1] 1.061135
```

### Movie and user effects

The next model used incorporates a movie effect (some movie are inherently better than others), and a user effect (some users tend to love everything they see, and therefore, while others are jaded critics and give lower ratings than average):

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

(The b in this notation stands for bias, aka effects)

The movie effect can be found by taking the difference of the avg rating for all movies, and the individual rating for each movie. Better than avg movies will have positive residuals, while worse than avg movies will have negative residuals.

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

```r
b_i <- train %>% group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))
```

In order to test the model this effect is then incorporated into the train and test data frame as such:

```r
train <- train  %>% left_join(b_i, by = "movieId")
test <- test  %>% left_join(b_i, by = "movieId")
```

Which gives us the following RMSE:

```r
RMSE(test$rating, mu + test$b_i)
```

```
## [1] 0.9441568
```

The user effect is incorporated in essentially the same way, except that the movie effect is also subtracted to further reduce variation:

```r
b_u <- train %>% group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))
train <- train  %>% left_join(b_u, by = "userId")
test <- test  %>% left_join(b_u, by = "userId")
```

The user effect and movie effect combined give us an RMSE of:

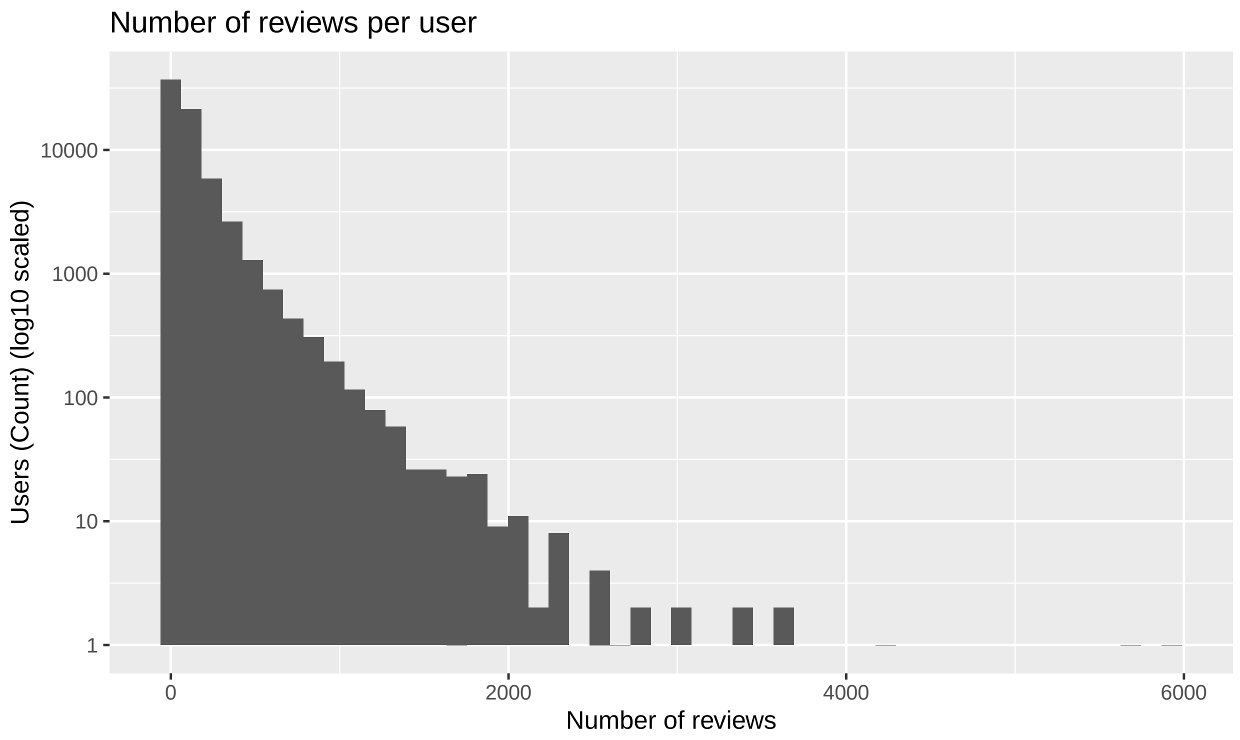
```r
RMSE(test$rating, mu + test$b_u + test$b_i)
```

```
## [1] 0.8659736
```

**Regularization**

The user and movie effects model can further be improved through regularization. Many of the movies in the dataset are quite obscure, and have only received one or two ratings; this makes their sample average rating extremely variable, and unrepresentative of the true average rating. The current model does not account for this, and the movie effect from movies with few ratings is likely to be overepresented. The same applies for users, as some have only rated 20 movies, while others thousands.

```
train %>% group_by(userId) %>% summarise(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 50) +
  scale_y_log10() +
  ylab("Users (Count) (log10 scaled)") +
  xlab("Number of reviews") +
  ggtitle("Number of reviews per user")
```



Regularization helps us account for this variation by adding a penalty $\lambda$ to large estimates of effects that come from small sample sizes.

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

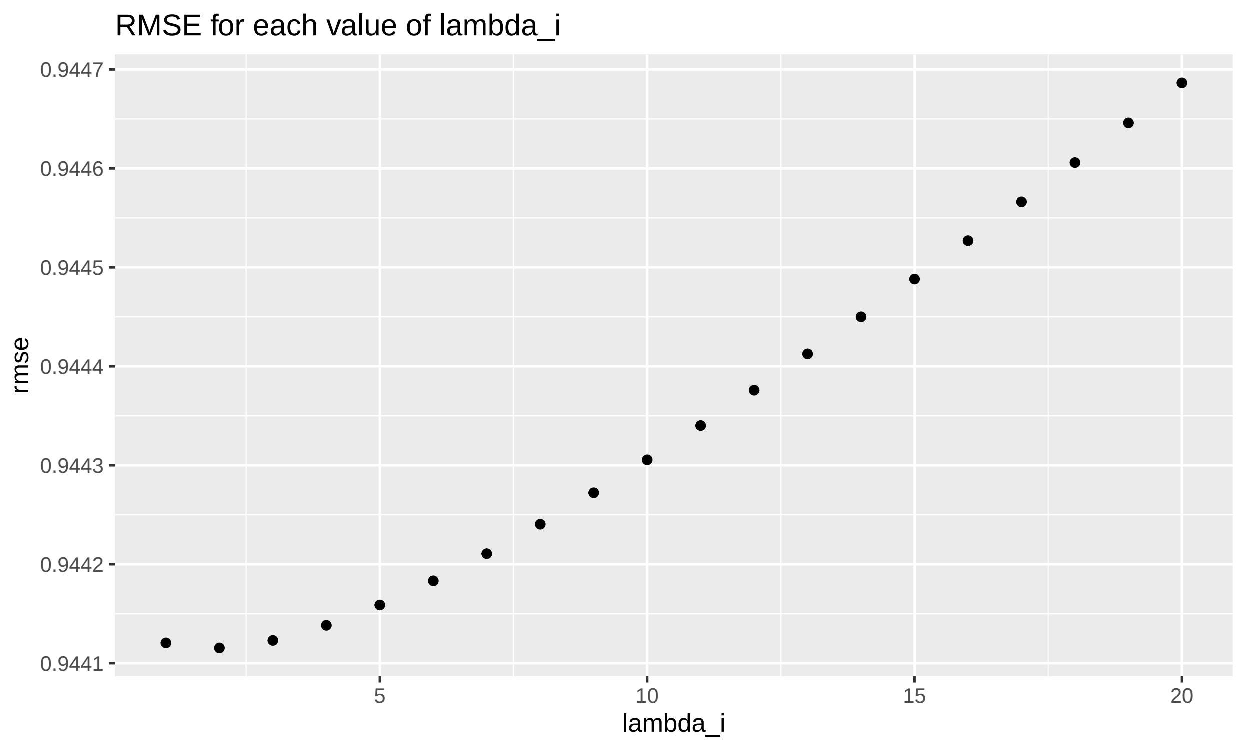
As $\lambda$ increases small sample sizes are further penalized.

In order to find the optimal value for $\lambda_i$, the following function is created, which finds the RMSE on the test set after training on the train set when given a specific $\lambda$ :

```
reg_RMSE_b_i <- function(l_i){
  b_i_reg <- train %>% group_by(movieId) %>%
    summarise(b_i_reg = sum(rating - mu)/(n() + l_i))
  test_r <- test %>% left_join(b_i_reg, by = "movieId")
  data.frame(rmse =  RMSE(test_r$rating, mu + test_r$b_i_reg), l_i = l_i)
}
```

$\lambda s$ 1 through 20 are tested :

```
lambdas <- 1:20
reg_rmses_b_i <- map_df(lambdas, function(x) reg_RMSE_b_i(x))
#plot results
reg_rmses_b_i %>% ggplot(aes(l_i, rmse)) + geom_point() +
  ggtitle("RMSE for each value of lambda_i") + xlab("lambda_i")
```

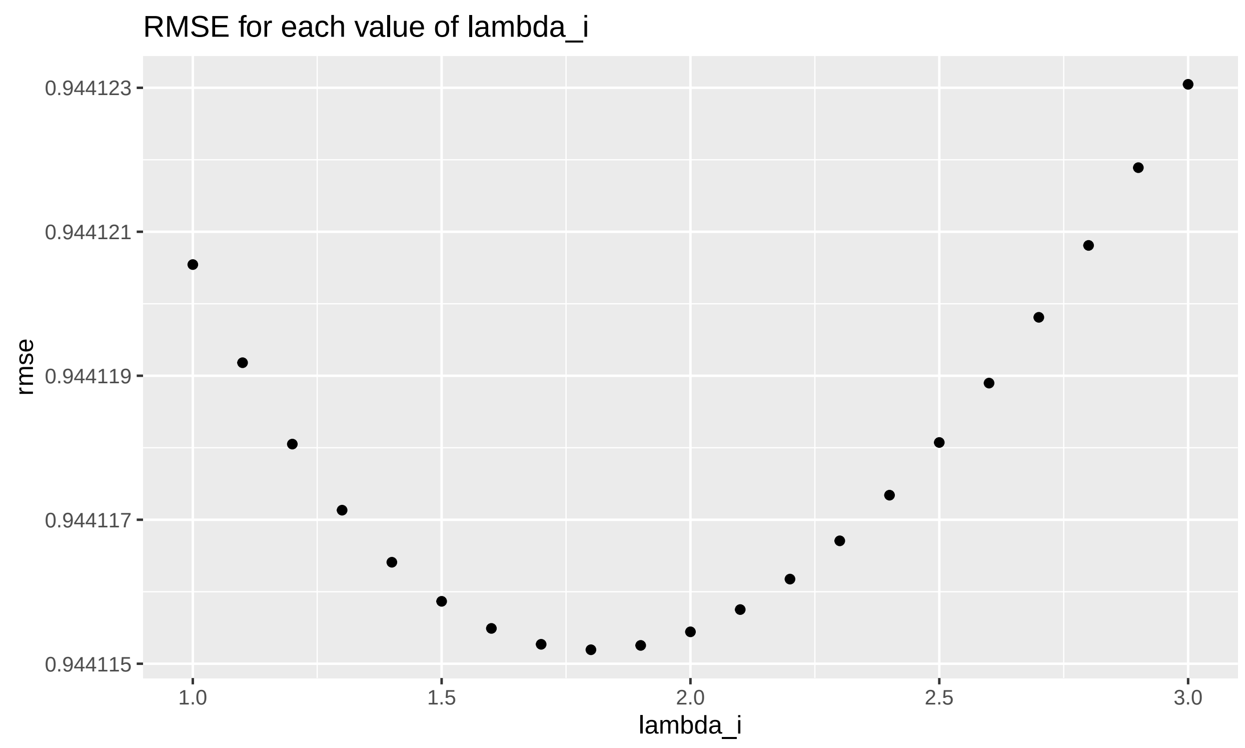


```
knitr::kable(reg_rmses_b_i[which.min(reg_rmses_b_i$rmse),])
```

| | rmse | l_i |
|---|---|---|
| 2 | 0.9441154 | 2 |

$\lambda s$ 1 through 3, with intervals of 0.1 are then tested to further narrow down $\lambda_i$ :

```
lambdas <- seq(1,3,.1)
reg_rmses_b_i <- map_df(lambdas, function(x) reg_RMSE_b_i(x))
reg_rmses_b_i %>% ggplot(aes(l_i, rmse)) + geom_point() +
  ggtitle("RMSE for each value of lambda_i") + xlab("lambda_i") #plot results
```



```
knitr::kable(reg_rmses_b_i[which.min(reg_rmses_b_i$rmse),])
```

|   | rmse | l_i |
|---|---|---|
| 9 | 0.9441152 | 1.8 |

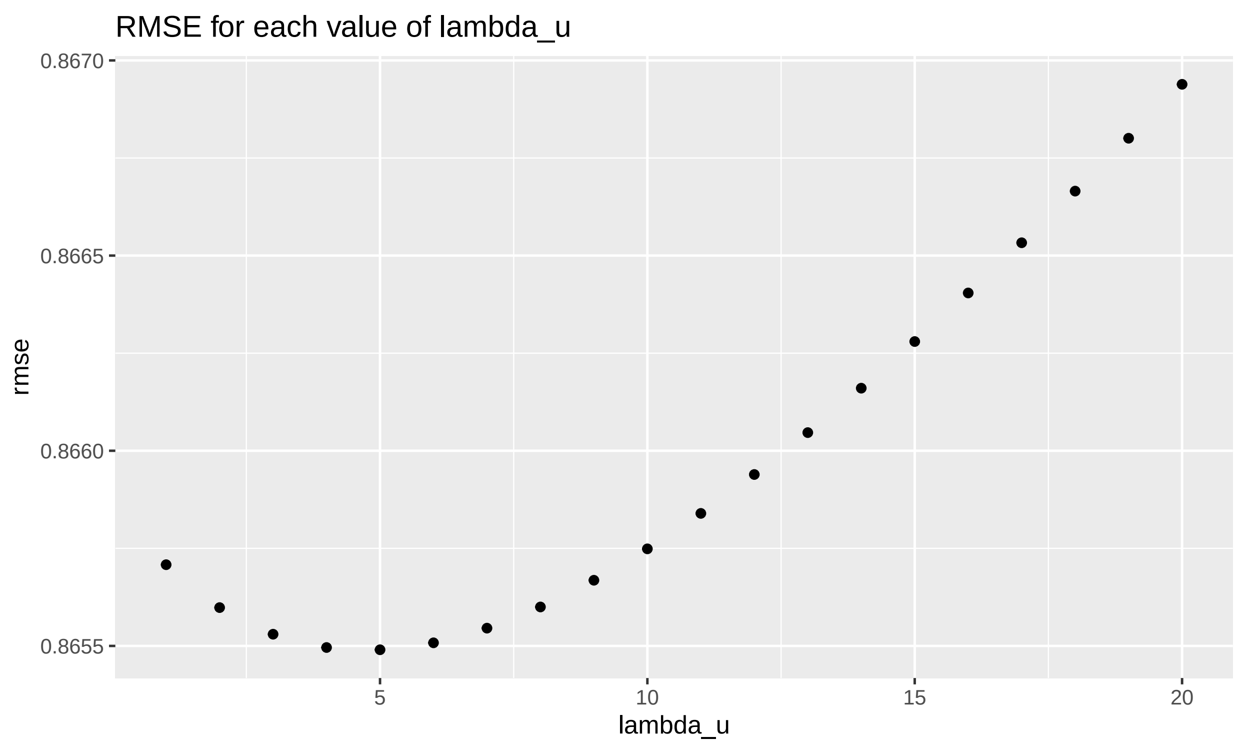
```
l_i <- reg_rmses_b_i[which.min(reg_rmses_b_i$rmse),]$l_i
```

To find the optimal $\lambda_u$, a new function is defined, which takes $\lambda_i$ and $\lambda_u$ as inputs:

```
reg_RMSE <- function(l_i, l_u){
  b_i_reg <- train %>% group_by(movieId) %>%
    summarise(b_i_reg = sum(rating - mu)/(n() + l_i))
  test_r <- test %>% left_join(b_i_reg, by = "movieId")
  train_r <- train %>% left_join(b_i_reg, by = "movieId")
  b_u_reg <- train_r %>% group_by(userId) %>%
    summarise(b_u_reg = sum(rating - mu - b_i_reg)/(n() + l_u))
  test_r <- test_r %>% left_join(b_u_reg, by = "userId")
  data.frame(rmse =  RMSE(test_r$rating, mu + test_r$b_i_reg + test_r$b_u_reg),
             l_i = l_i , l_u = l_u)
}
```

$\lambda s$ 1 through 20 are once again tested for $\lambda_u$, and 2.3 is used for $\lambda_i$:

```
lambdas <- 1:20
reg_rmses <-  map_df(lambdas, function(x) reg_RMSE(l_i, x))
reg_rmses %>% ggplot(aes(l_u, rmse)) + geom_point() +
  ggtitle("RMSE for each value of lambda_u") + xlab("lambda_u") #plot results
```

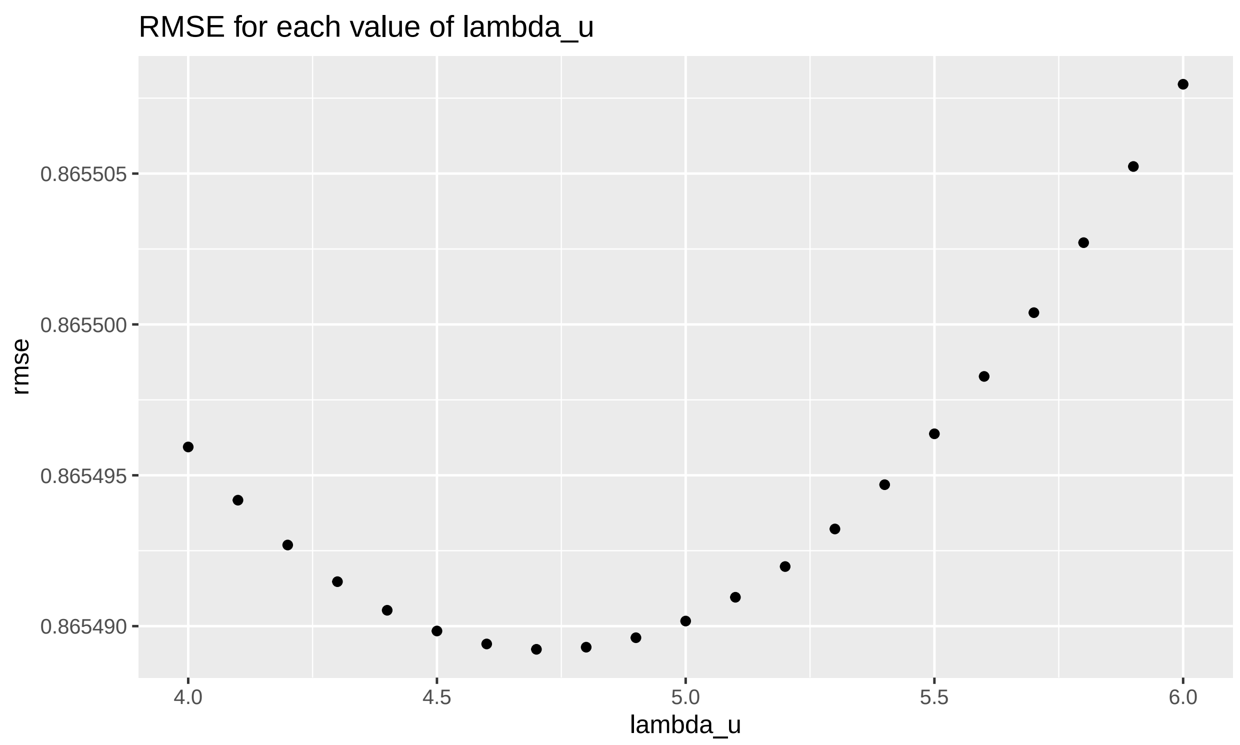


```
knitr::kable(reg_rmses[which.min(reg_rmses$rmse),])
```

| | rmse | l_i | l_u |
|---|---|---|---|
| 5 | 0.8654902 | 1.8 | 5 |

Narrowing down $\lambda_u$ in the same way as before gives us an optimal value of 4.7:

```
lambdas <- seq(4,6, by = .1)
reg_rmses <-  map_df(lambdas, function(x) reg_RMSE(l_i, x))
reg_rmses %>% ggplot(aes(l_u, rmse)) + geom_point() +
  ggtitle("RMSE for each value of lambda_u") + xlab("lambda_u")
```



```
l_u <- reg_rmses[which.min(reg_rmses$rmse),]$l_u
```

And our regularized model now has an estimated RMSE of:

```
reg_RMSE(l_i, l_u)
```

```
##        rmse l_i l_u
## 1 0.8654892 1.8 4.7
```

**Matrix Factorization:**

**What is it?**

In order to further increase the accuracy of our model, matrix factorization can be used.
Matrix factorization essentially works by **finding similarities** in movies and users not accounted for by the user and movie effects, these similarities are clumped together into groups called *latent factors*.

-There are **user** factors, which are stored in a **"P" matrix**. (With each row representing a factor, and each column representing a user)

-And there are **item (movie) factors**, which are stored in a **"Q" matrix**. (With each row representing a factor, and each column representing a movie)

Each factor in the **Q (movie) matrix** attempts to find a **specific attribute/characteristic** that would correspond with many movies, such as "blockbuster" which on one end of the spectrum would include blockbuster movies such as "Avengers" and would include indie movies such as the "Blair Witch project" on the other end. Movies in between might receive a 0, while blockbusters would receive a positive number, and indie movies a negative. There will be many different types of factors, some may rely on genre such as a "fantasy" factor, or a particular actor in a set of movie, maybe a "Dicaprio" factor. Some factors may be completely uninterpretable, and rely on some obscure quirks of random movies.

Factors in the **P (user) matrix** will record **how much interest each user has in any given movie factor**. For example a user who hates blockbusters, loves fantasy, and doesn't care about Dicaprio might get a positive score on the blockbuster factor, a negative for the fantasy factor, and a zero for the Dicaprio factor.

Taking the dot product $pT_u * q_i$ of the two matrices results in the **R (Residual) Matrix**. This matrix captures each user's intersest in a particular movie's characteristics (latent factors). This is the estimate of the user rating for each movie $Y_{u,i}$.

The goal of Matrix Factorization is to estimate ratings as accurately as possible, i.e. minimize the cost function** for P and Q: (Chin, Zhuang, et al. 2015a, 2015b)

$$\min_{P,Q} \sum_{(u,i)\in S} \left[ f(p_u, q_i; r_{u,i}) + \mu_P ||p_u||_1 + \mu_Q ||q_i||_1 + \frac{\lambda_P}{2} ||p_u||_2^2 + \frac{\lambda_Q}{2} ||q_i||_2^2 \right]$$

Where S is the set of all observed ratings, $f$ is the loss function (RMSE in our case) $\mu_P$, $\mu_Q$, $\lambda_P$ and $\lambda_Q$ are the regularization (penalty) parameters to avoid overfitting.

**note that there are other possible representations of cost function, especially due to different ways of using regularization, here is another possible approach with only one regularization parameter:

$$f(cost) = \sum_{r_{u,i}\in S} (pT_u * q_i - Y_u, i)^2 + \lambda * (\sum_u ||p_u||^2 + \sum_i ||q_i||^2)$$

**Implementation:**

To implement matrix factorization into our algorithm, the *recosystem* r package can be used. This package makes use of the LIMBF library developed by Yu-Chin Juan, Wei-Sheng Chin, Yong Zhuang, Bo-Wen Yuan, Meng-Yuan Yang, and Chih-Jen Lin (http://www.csie.ntu.edu.tw/~cjlin/libmf/).

This package uses parallel Stochastic Gradient Descent (SGD) in order to minimize the cost function.

The model including matrix factorization now becomes:

$$\mathbf{Y_{u,i}} = \mu + b_i + b_u + \sum_{m=1}^{k} (p_{u,m}\, q_{i,m}) + \epsilon_{u,i}$$

Where k is the number of latent factors. Each latent factor, in a way, can be thought of as an additional baseline effect.

To start the matrix factorization, we first need to add our previous baseline effects into the train and test sets, aswell as the rating minus effects into the train set:

```
b_i_reg <- train %>% group_by(movieId) %>%
  summarise(b_i_reg = sum(rating - mu)/(n() + l_i))
test <- test %>% left_join(b_i_reg, by = "movieId")
train <- train %>% left_join(b_i_reg, by = "movieId")
b_u_reg <- train %>% group_by(userId) %>%
  summarise(b_u_reg = sum(rating - mu - b_i_reg)/(n() + l_u))
train <- train %>% left_join(b_u_reg, by = "userId") %>%
  mutate(rating_minus_effects = rating - b_i_reg - b_u_reg - mu)
test <- test %>% left_join(b_u_reg, by = "userId")
```

We then need to define the training and testing data and define r as a recosystem (RecoSys) object:

```r
train_data <- data_memory(user_index = train$userId, item_index = train$movieId,
                          rating = train$rating_minus_effects, index1 = T)
test_data <- data_memory(user_index = test$userId, item_index = test$movieId, index1 = T)
r <- Reco()
class(r)[1]
```

```
## [1] "RecoSys"
```

The model parameters are then tuned using 5-fold cross validation in the “tune” function; they are tuned independently (one by one) to save time since our computing power is somewhat limited.

```r
opts <- r$tune(train_data = train_data,
               opts = list(dim = 45, costp_l1 = 0,
                           costp_l2 = .01, costq_l1 = 0, costq_l2 = .2,
                           lrate = .1, verbose = T, nfold = 5))
# Best parameters:
# dim c(seq(5,35,5), seq(32.5,45,2.5)) = 45
# costp_l1 (c(0,.01,.05,.1,.15,.2)) = 0
# costp_l2 (c(0,.01,.05,.1,.15,.2)) = .01
# costq_l1 (c(0,.01,.05,.1,.15)) = 0
# costq_l2 (c(0,.01,.05,.1,.15,.2, .25,.3)) = .2
# lrate = c(.01, .05, .1, .15, .2) = .1
```

The algorithm is then trained using the best parameters acquired from tuning:

```r
r$train(train_data = train_data,
        opts = c(niter = 40, dim = 45, costp_l1 = 0, costp_l2 = 0.01,
                 costq_l1 = 0, costq_l2 = .2, lrate = .1, verbose = FALSE))
```

With the model trained, we can make our predictions on the test set:

```r
y_hat_sgd <- (r$predict(test_data, out_pred = out_memory())
              +mu + test$b_i_reg + test$b_u_reg)
```

And we can therfore get our final estimated RMSE:

```r
RMSE(test$rating, y_hat_sgd)
```

```
## [1] 0.7913268
```

## Results

With our final model chosen:

$$\mathbf{Y_{u,i}} = \mu + b_i + b_u + \sum_{m=1}^{k} (p_{u,m}\, q_{i,m}) + \epsilon_{u,i}$$

We are ready to test it against the validation set.

In order to take advantage of the total size of the edx dataset, we will retrain the model on the full edx set:

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

# Movie and user effects
reg_b_i <- edx %>% group_by(movieId) %>%
  summarise(reg_b_i = sum(rating - mu)/(n() + l_i))
edx <- edx %>% left_join(reg_b_i, by = "movieId")
reg_b_u <- edx %>% group_by(userId) %>%
  summarise(reg_b_u = sum(rating - mu - reg_b_i)/(n() + l_u))
edx <- edx %>% left_join(reg_b_u, by = "userId") %>%
  mutate(rating_minus_effects = rating - reg_b_i - reg_b_u - mu)

# Matrix factorization
edx_data <- data_memory(user_index = edx$userId, item_index = edx$movieId,
                        rating = edx$rating_minus_effects, index1 = T)

r_final <- Reco()
r_final$train(train_data = edx_data,
              opts = c(niter = 40, dim = 45, costp_l1 = 0, costp_l2 = 0.01,
                       costq_l1 = 0, costq_l2 = .2, lrate = .1, verbose = F))
```

Now that the model has been retrained on the full edx set, it's time for the final test on the validation set (which hasn't been used in any part of our training/decisionmaking, this is the first time it is used).

First we join the b_i_reg and b_u_reg effects as columns on the validation set (note: this does not affect the dataset in any way, it simply adds a column for the corresponding b_i and b_u regularized effects) :

```r
validation <- validation %>% left_join(reg_b_i, by = "movieId") %>%
  left_join(reg_b_u, by = "userId")
```

Then we can define the validation data* and make our final predictions:

```r
validation_data <- data_memory(user_index = validation$userId,
                               item_index = validation$movieId,
                               rating = NULL, index1 = T)

final_predictions <-  mu + validation$reg_b_i + validation$reg_b_u +
  r_final$predict(validation_data, out_pred = out_memory())
```

*rating is left null, therefore it is not used, only movie_Id and user_Id are used

And the final RMSE is:

```r
rmse_final <- RMSE(validation$rating, final_predictions)
rmse_final
```

```
## [1] 0.7875875
```

This result RMSE is a lot better than first expected, as it is 11.9916% smaller than the target RMSE, and 25.7788% smaller than the RMSE for the baseline guessing RMSE!

## Conclusion

While we were able to achieve a good RMSE, there are still many possible additions to the final model. For instance, the model does not currently make any use of the timestamp or given genres. More in depth descriptions of certain methods used (such as SGD) could also be beneficial.

Future improvements to the model could include additional baseline factors, aswell as more types of models, such as a potential nearest neighboors model.

## Citations:

-Chin, Wei-Sheng, Bo-Wen Yuan, Meng-Yuan Yang, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. 2015. "LIBMF: A Library for Parallel Matrix Factorization in Shared-Memory Systems."
https://www.csie.ntu.edu.tw/~cjlin/papers/libmf/libmf_open_source.pdf.
-Chin, Wei-Sheng, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. 2015a. "A Fast Parallel Stochastic Gradient Method for Matrix Factorization in Shared Memory Systems." ACM TIST.
http://www.csie.ntu.edu.tw/~cjlin/papers/libmf/libmf_journal.pdf.
———-. 2015b. "A Learning-Rate Schedule for Stochastic Gradient Methods to Matrix Factorization." PAKDD.
http://www.csie.ntu.edu.tw/~cjlin/papers/libmf/mf_adaptive_pakdd.pdf.
-F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

*Created as a capstone project for Harvardx's proffessional certificate in Data Science on Edx.*