MovieLens Project

RenDerMCh

3/21/2020

Contents

INTRODUCTION

As per the instructions of this part of the Harvard Data Science Professional Certificate program we were asked to create a recommendation system for movie recommendations. Similar to the recommendation system asked by Netflix during the Neflix challenge.

We were given a data set described in further sections and provided and indication of desired result (RMSE < 0.8649).

This report describes the data, methods and results of my grasping of this project.

Provided data set description

Harvard team provided us with data set as well as initial code diving the data set into two parts: 1.) edx - set intended for training of the models 2.) validation - set intended for application of the chosen final model and calculating Root Mean Square Error (RMSE)

The initial data set comes from GroupLens and their MovieLens dataset. The MovieLens dataset included 10000054 observations (ratings). We are including the initial the first ratings to demonstrate the data structure in Table 1.

Table 1: MovieLens data set - first 6 ratings

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

Splitting MovieLens data set into edx and validation sets

As the same data set is to be used for training our model as well as applying it to an "unknown" dataset, it is needed to split the MovieLens dataset into two part.

edx data set - will be used for training and testing the models and is to contain 90% of the data of MovieLens set validation set - will represent an "unknown" dataset on which we will apply our chosen model and calculate its error, ensuring that validation set does not contain movies and users that are not in the edx (else we would introduce NA's into the models)

Creation of the data partition will involve setting.seed - so that the data remains the same for all that run the code so that the values are verifiable.

Code used:

```r
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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```r
edx <- rbind(edx, removed)
```

Once split we ended up with edx data set with **9000055 observations** and validation set with 999999 observations/ratings. Together they have **10000054 observations/ratings** which is identical with the number of ratings in the MovieLens dataset described above.

ANALYSIS AND METHODS

Preping data sets for analysis

As we have to build and test our models only on the edx set, we have decided to further split the edx data set into a training set (train_set) and test set (test_set) this way, we can built the models on the training set and use the test set to fine tune and test the models, as to choose the final model. Again selecting a seed of 10221 to maintain results for all running the code.

```r
set.seed(10221, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating,
  times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

Also since we are creating the data sets for training we will need to verify that the test set contains all movies and all users that the are in training set as well, as not to introduce NA's into the models. Since we are doing this now we will also modify the sets to include a date column creating from timestamp variable and use this opportunity to also test that the test set only includes ratings that have the dates in train set as well.

```r
train_set<- train_set %>%
  mutate(dates=(as.Date(as.POSIXct(train_set$timestamp, origin="1970-01-01"))))
test_set<- test_set %>%
  mutate(dates=(as.Date(as.POSIXct(test_set$timestamp, origin="1970-01-01"))))
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId") %>%
  semi_join(train_set, by = "dates")
```

Once split we ended up with training data set with **8100048 observations** and testing set with **899990 observations/ratings**. Together they have **9000038 observations/ratings** which is not identical with the number of ratings in the edx dataset described above, but for the purpose of the training this is irrelevant as once the proper model is chosen it will be retrained on the whole edx data set.

Further we have to create a metric for testing of our models. We will be using root mean squared error (RMSE), and is calculated as follows in our code:

```r
RMSE <- function(true, predicted){sqrt(mean((true - predicted)^2))}
```

Basic exploratory data analysis

Doing basic exploratory data analysis on the training set (first 10 ratings shown in Table 2). We have observed that the available parameters are:

- **userId** - identifying the user who submitted the rating
- **movieId** & **title** - identifying the movie
- **timestamp** - timestamp of the rating, from which we have extracted **dates** containing the date of the rating
- **genres** - identifying all the genres that the movie belongs to

Table 2: Training data set - first 10 ratings

userId	movieId	rating	timestamp	title	genres	dates
1	122	5	838985046	Boomerang (1992)	Comedy|Romance	1996-08-02
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller	1996-08-02
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1996-08-02
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi	1996-08-02
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1996-08-02
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy	1996-08-02
1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War	1996-08-02
1	362	5	838984885	Jungle Book, The (1994)	Adventure|Children|Romance	1996-08-02
1	364	5	838983707	Lion King, The (1994)	Adventure|Animation|Children|Drama|Musical	1996-08-02
1	370	5	838984596	Naked Gun 33 1/3: The Final Insult (1994)	Action|Comedy	1996-08-02

The basic information gathered in the basic exploratory data analysis

- number of users that rated movies within training set is *69878*
- number of movies that had ratings submitted is *10661*
- if all users submitted ratings for all movies that would result in *744969358* observations/ratings
- we only have *8100048* observations/ratings in the training set which presents only a proportion of *0.011* of the total possible observations/ratings (number users multiplied by number movies)

Therefore the scope of this project is to model the missing 98.9% of the data and then apply it to the testing set to verify it's accuracy.

Further information about the movies are absent and for the purposes of this project we will not look to obtain more data. If we were to extend the data available, we would probably consider the following (to extend the data to further enhance the model and investigate the influence of this data on the models):

- *main cast* - the assumption being that main star of the movie has influence on the ratings as different users like different starts and reflect that in the ratings
- *budget* - although probably strongly correlated with number of ratings - it might still be worth exploring if movie budget has further influence on the accuracy of the models
- *movie release date* - we could extract this data from the movie title and explore the influence of age of a movie on the ratings assumption being that the older the movie the lower the rating due to the movie effects quality, etc.

MODELS

The easiest model is to predict the average of the ratings for all "unrated" movies and users.

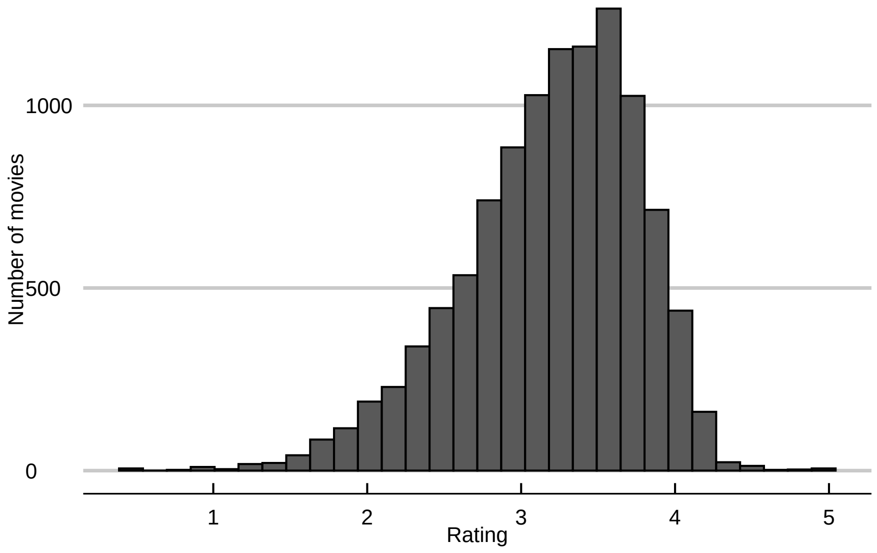
```
mu <- mean(train_set$rating)
```

But from the above basic exploratory data analysis we know we should be able to do better, so we will not calculate the loss function or discuss the plain average any further, but we going to move to a bit more complicated models.

Modeling movie effect

Since we know that movies are generally rated differently, good movies are rated higher and bad movies are rated lower as can be seen in Figure 1. We will use this as our first model. We will assign all ratings for movie i in our model the average rating for movie i.

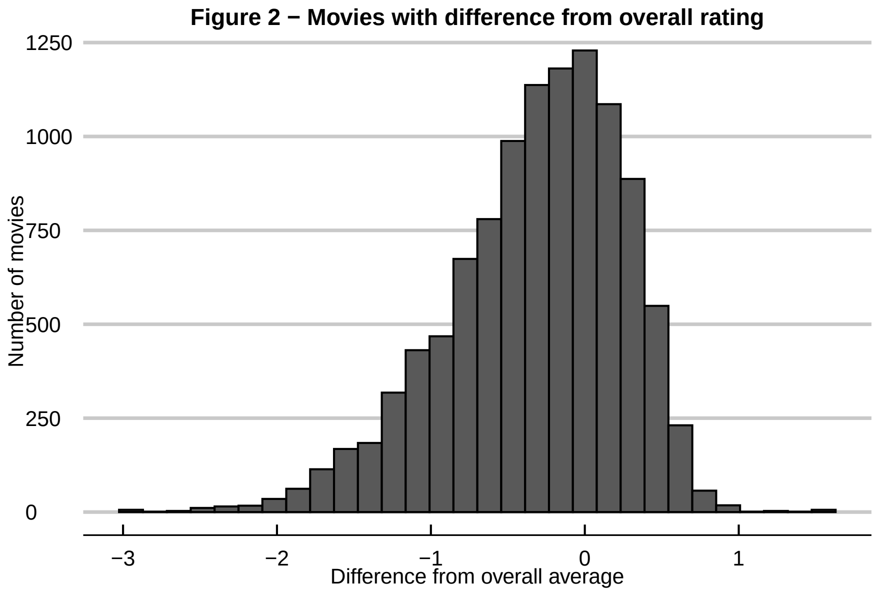
Figure 1 – Number of movies with various average ratings



As we are going to be extending the use of this model further on, we will not use a direct average for each movie, rather we will calculate the difference of average rating for movie *i* from the overall average *mu*.

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

We repeat the above figure, but this time displaying the difference from overall average instead of the average rating of each movie.



Now we use the obtained differences from overall average from the training set to predict the ratings in the test set. And calculate the loss function (RMSE) and display the result.

```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
model_M_rmse <- round(RMSE(predicted_ratings, test_set$rating),digits=5)
rmse_results <- data_frame(method = "Movie Effect", RMSE_train = model_M_rmse)
kable(rmse_results,caption = "Model Results",booktabs=TRUE) %>%
  kable_styling(full_width = F,
                latex_options = c("striped","hold_position"))
```

Table 3: Model Results

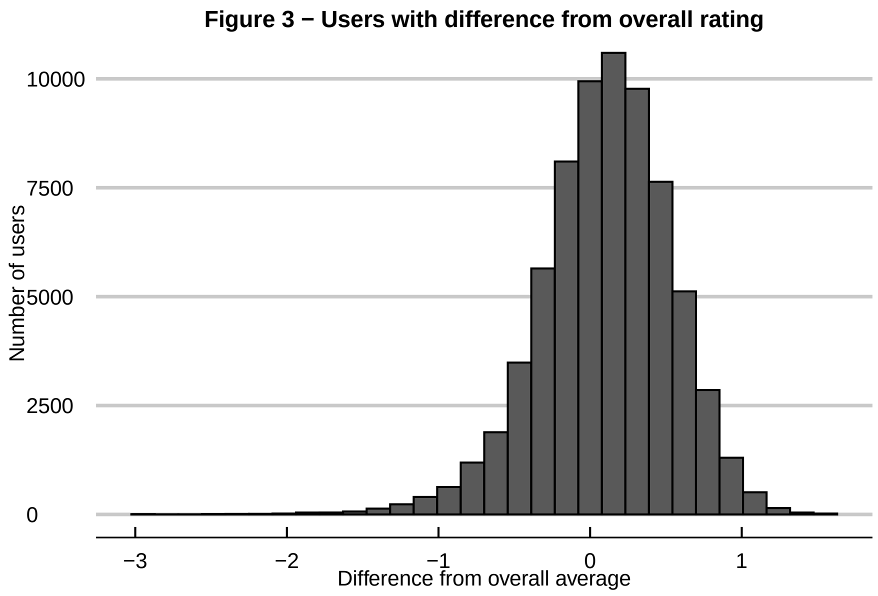
method	RMSE_train
Movie Effect	0.94452

We should do better. So the next variable we are going to take into account it the users.

Modeling user effect

Each user has different tastes and standards, therefore we assume that each user can rate differently (i.e. we can have grumpy users that complain a lot and therefore rate lower overall). We will explore this effect. We will assign all ratings for user *u* in our model the average rating for user *u*. In Figure 3 we show that different users have different difference from the overall average ratings.

```r
user_avgs <- train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu))
```



Now we use the obtained differences from overall average from the training set to predict the ratings in the test set. And calculate the loss function (RMSE) and display the result.

```r
predicted_ratings <- mu + test_set %>%
  left_join(user_avgs, by='userId') %>%
  .$b_u
model_U_rmse <- round(RMSE(predicted_ratings, test_set$rating),digits=5)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="User Effects",
                                     RMSE_train = model_U_rmse))
kable(rmse_results,caption = "Model Results",booktabs=TRUE) %>%
  kable_styling(full_width = F,
              latex_options = c("striped","hold_position"))
```

Table 4: Model Results

method	RMSE_train
Movie Effect	0.94452
User Effects	0.97917

We see that the user effect has a higher error (larger value of the loss function - RMSE) and therefore is less suitable then the movie effect, but what if we combine the two together.

Modeling movie + user effect

We build a model using both previously obtained effects on the ratings. We combine the movie effect and user effect into one training model. We use the difference from average of movie i obtained in the movie effect model to recalculate the user effect for each user u but already taking into account the movie effect.

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

Now we use this newly trained model to predict the ratings in the test set. And calculate the loss function (RMSE) and display the result.

```r
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
model_MU_rmse <- round(RMSE(predicted_ratings, test_set$rating),digits=5)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects",
                                     RMSE_train = model_MU_rmse))
kable(rmse_results,caption = "Model Results",booktabs=TRUE) %>%
  kable_styling(full_width = F,
                latex_options = c("striped","hold_position"))
```

Table 5: Model Results

method	RMSE_train
Movie Effect	0.94452
User Effects	0.97917
Movie + User Effects	0.88592

We see that we have obtained a much better result by combining these to effect than from either of them alone. But we should be able to do better.

Further data exploration

If we look deeper into the training set we can see that the users and movies that have the largest difference from the average have the number of ratings far below average.

```r
top_users<-train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu),n=n()) %>%
  top_n(b_u,n=10) %>% arrange(desc(b_u))
kable(head(top_users,10),caption = "Top rating users",booktabs=TRUE) %>%
  kable_styling(full_width = F,
                latex_options = c("striped","hold_position"))
```

In Table 6 we can see that the top rating users (only first 10 lines displayed) have an average of *23.95* of ratings while the overall average of ratings per user is *115.92*. Which is far below the overall average of number of ratings per user. NOTE: The users displays have b_u of 1.48749 - which is the difference from average mu and $b\_u + mu = 5$ - a perfect rating.

```r
bot_movies<-train_set %>%
  group_by(title) %>%
  summarize(b_i = mean(rating - mu),n=n()) %>%
  top_n(-b_i,n=10) %>% arrange(b_i)
kable(head(bot_movies,10),caption = "Bottom rated movies",booktabs=TRUE) %>%
```

Table 6: Top rating users

userId	b_u	n
1	1.487494	18
7984	1.487494	16
11884	1.487494	14
13027	1.487494	25
13513	1.487494	17
13524	1.487494	17
15575	1.487494	24
18965	1.487494	44
22045	1.487494	13
26308	1.487494	16

```
  kable_styling(full_width = F,
                latex_options = c("striped","hold_position"))
```

Table 7: Bottom rated movies

title	b_i	n
Accused (Anklaget) (2005)	-3.012506	1
Besotted (2001)	-3.012506	1
Confessions of a Superhero (2007)	-3.012506	1
Hi-Line, The (1999)	-3.012506	1
Under the Lighthouse Dancing (1997)	-3.012506	1
War of the Worlds 2: The Next Wave (2008)	-3.012506	1
SuperBabies: Baby Geniuses 2 (2004)	-2.726792	49
Hip Hop Witch, Da (2000)	-2.691078	14
Disaster Movie (2008)	-2.601792	28
From Justin to Kelly (2003)	-2.586792	175

In Table 7 we can see that the bottom rated movies (only last 10 lines displayed) have an average of *27.2* of number of ratings while the overall average of the number of ratings per movie is *759.78*. Which is far below the overall average of number of ratings per movie

The lower the number of ratings per user or per movie gives us less options to estimate the average per user or per movie. The less data we have about something the worse we understand it - the less data the more it could be influenced just by 1 bad rating … Therefore we should move the predictions based on this lower number of ratings more towards the overall average.

Regularized Movie Effect

We can move the predictions of movies with lower number of ratings more towards the overall average - we can penalize them for having a lower number of ratings. We also need to find how much we should penalize them. We will use lambda as the penalty and we can optimize lambda.

```
lambdas <- seq(0, 10, 0.25)
just_the_sum <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
```

```
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
lambda_m<- lambdas[which.min(rmses)]
```

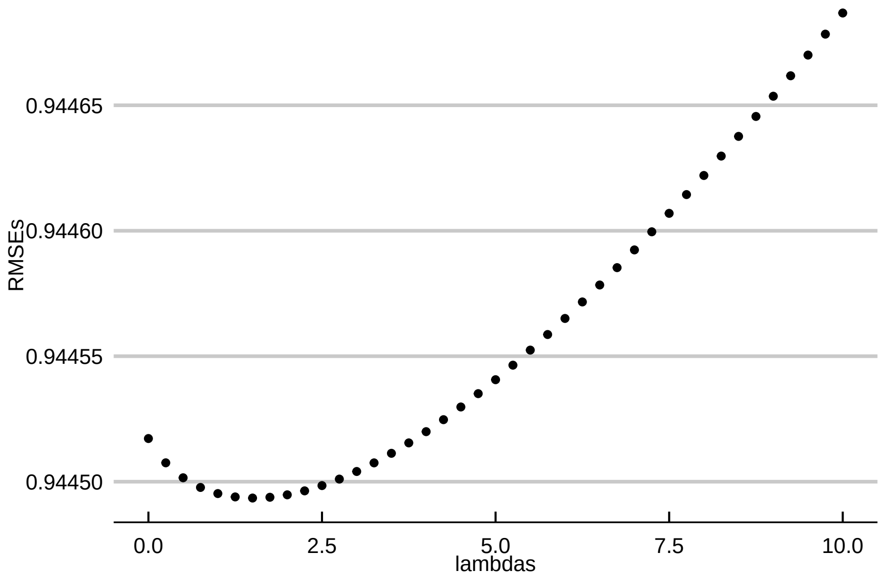
We basically run the same code as for the Movie Effect, but with two differences - 1.) we do not include a simple average into b_u but we increase the number of observations/ratings by lambda and 2.) we run the code in a loop to test for all different lambdas. Once done we select the lambda that optimized the model on the test set and we can display the results:

Table 8: Model Results

method	LAMBDA	RMSE_train
Movie Effect		0.94452
User Effects		0.97917
Movie + User Effects		0.88592
Regularized Movie Effect	1.5	0.94449

We can see from the results that with *lambda = 1.5* the RMSE decreases slightly as opposed to the Movie Effect without regularization.

We can plot the rmses vs. the lambdas to see that *lambda = 1.5* optimizes the model.



Regularized User Effect

We can move the predictions of user ratings with lower number of ratings more towards the overall average - we can penalize them for having a lower number of ratings. We also need to find how much we should penalize them. We will use lambda as the penalty and we can optimize lambda.

```r
lambdas <- seq(0, 10, 0.25)
just_the_sum <- train_set %>%
  group_by(userId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='userId') %>%
    mutate(b_u = s/(n_i+l)) %>%
    mutate(pred = mu + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
lambda_u<- lambdas[which.min(rmses)]
```

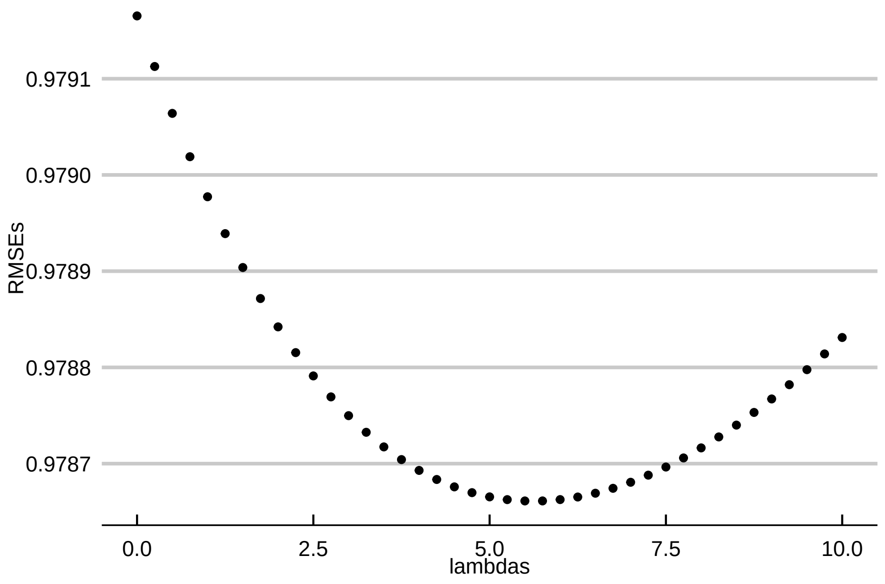
We basically run the same code as for the User Effect, but with two differences - 1.) we do not include a simple average into b_i but we increase the number of observations/ratings by lambda and 2.) we run the code in a loop to test for all different lambdas. Once done we select the lambda that optimized the model on the test set and we can display the results:

Table 9: Model Results

method	LAMBDA	RMSE_train
Movie Effect		0.94452
User Effects		0.97917
Movie + User Effects		0.88592
Regularized Movie Effect	1.5	0.94449
Regularized User Effect	5.5	0.97866

We can see from the results that with *lambda = 5.5* the RMSE decreases slightly as opposed to the User Effect without regularization.

We can plot the rmses vs. the lambdas to see that *lambda = 5.5* optimizes the model.



Now we can try to combine the Regularized Movie Effect and Regularized User Effect together

Regularized Movie + User Effect

We build a model using both previously obtained regularized effects on the ratings. We combine the regularized movie effect and regularized user effect into one training model. We use the difference from average of movie i obtained in the regularized movie effect model to recalculate the regularized user effect for each user u but already taking into account the regularized movie effect. We will use lambda as the penalty and we can optimize lambda by running the code in a loop to test for all different lambdas.

```r
lambdas <- seq(4, 6, 0.1)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
lamda_reg_mu <- lambdas[which.min(rmses)]
```

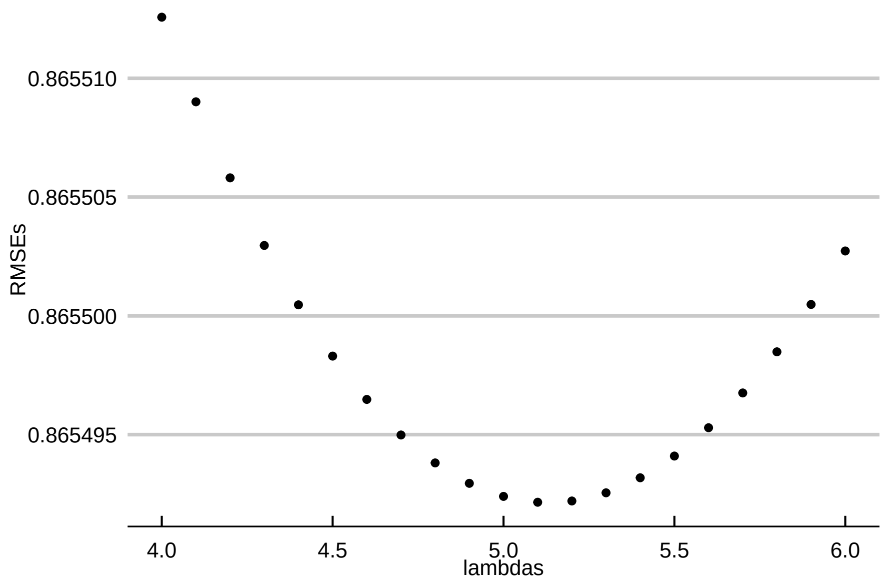
Once done we select the lambda that optimized the model on the test set and we can display the results:

Table 10: Model Results

method	LAMBDA	RMSE_train
Movie Effect		0.94452
User Effects		0.97917
Movie + User Effects		0.88592
Regularized Movie Effect	1.5	0.94449
Regularized User Effect	5.5	0.97866
Regularized Movie + User Effect Model	5.1	0.86549

We see that we have obtained a much better result by combining these two effect than from either of them alone and we also received a significant improvement as opposed to Movie + User Effect. The threshold that we would like achieve is still a bit further though - we will look for further variables to input into the model. The next one that is ready to go is the genre variable.

We can plot the rmses vs. the lambdas to see that *lambda = 5.1* optimizes the model.



Regularized Movie + User + Genre Effect

We build on the previous model, but this time around we also include the Genre Effect, which we also regularize to penalize the genres that have lesser numbers of movies. For the purpose of this project we do not split the genres into different columns, but we use them as is - this way the combination of genres is maintained in the model.

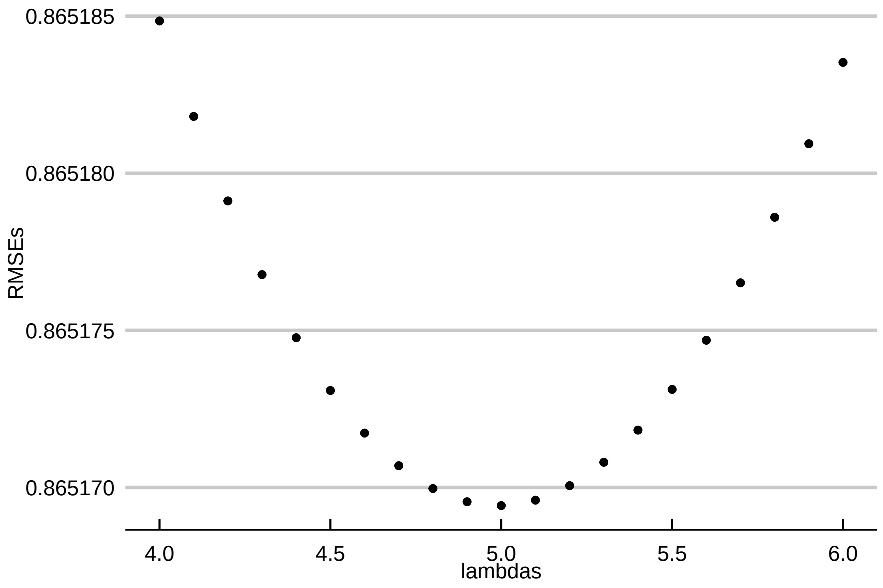
```
lambdas <- seq(4, 6, 0.1)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
lamda_reg_mug <- lambdas[which.min(rmses)]
```

Once done we select the lambda that optimized the model on the test set and we can display the results:

We can plot the rmses vs. the lambdas to see that $lambda = 5$ optimizes the model.

Table 11: Model Results

method	LAMBDA	RMSE_train
Movie Effect		0.94452
User Effects		0.97917
Movie + User Effects		0.88592
Regularized Movie Effect	1.5	0.94449
Regularized User Effect	5.5	0.97866
Regularized Movie + User Effect Model	5.1	0.86549
Regularized Movie + User + Genre Effect	5	0.86517



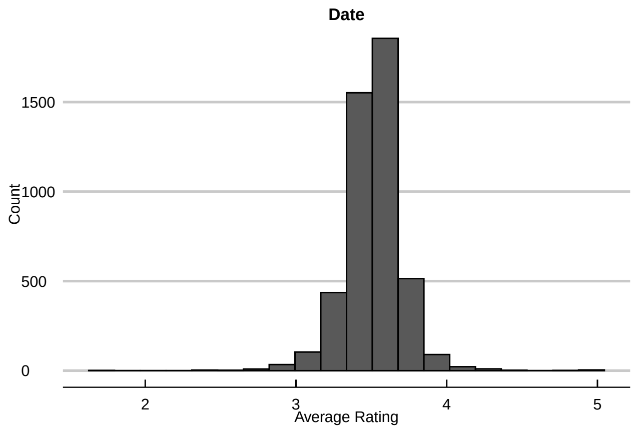
We have received further improvement on the previous model, but we would like to get below 0.8649 even with the training model, for the final model we expect and even a little bit lower RMSE as we will use the whole edx set for training not just 90% of it.

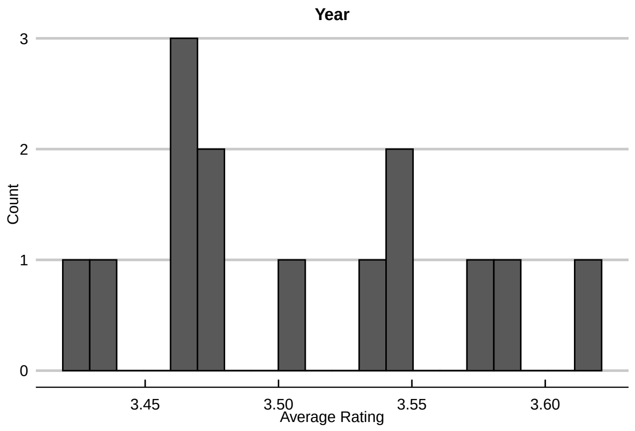
Further data exploration

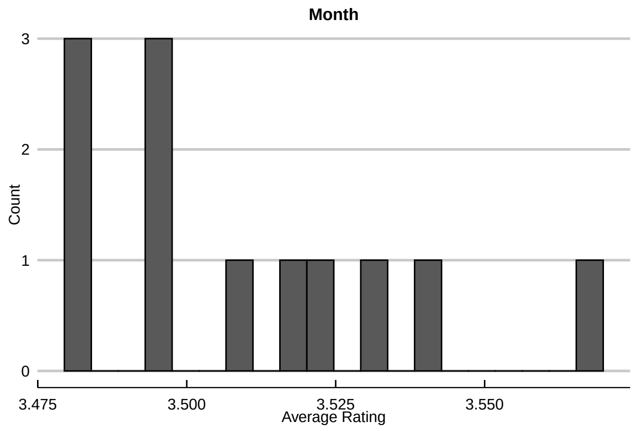
We will extract as much data from the timestamp as we think would be helpful, we already have the full Date, we will extract the Year, Month, Week, Week Day and Hour.

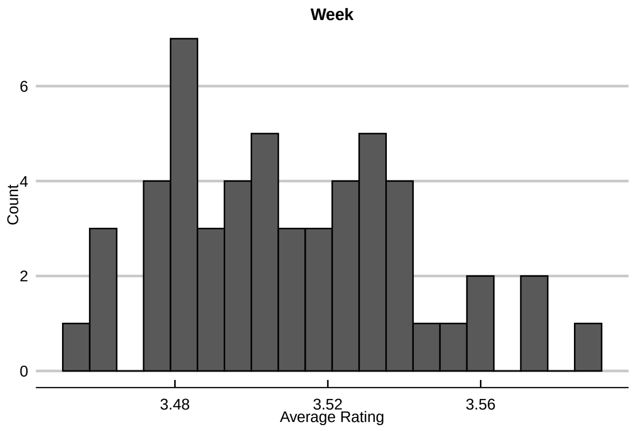
```
train_set2<- train_set %>%
  mutate(year=year(as.POSIXct(train_set$timestamp, origin="1970-01-01")),
         month=month(as.POSIXct(train_set$timestamp, origin="1970-01-01")),
         week=week(as.POSIXct(train_set$timestamp, origin="1970-01-01")),
         day=wday(as.POSIXct(train_set$timestamp, origin="1970-01-01")),
         hour=hour(as.POSIXct(train_set$timestamp, origin="1970-01-01")))
```

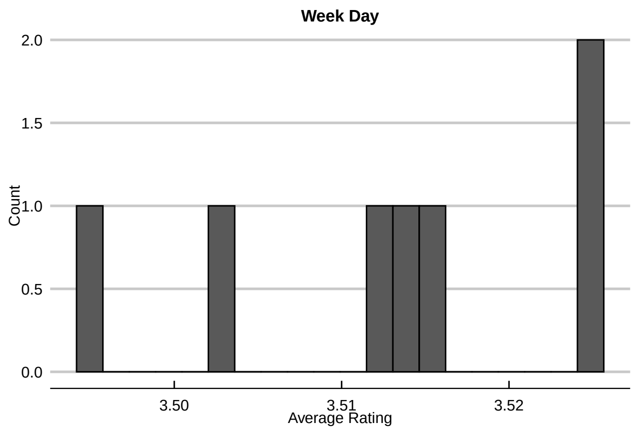
And we plot the histogram of these variables

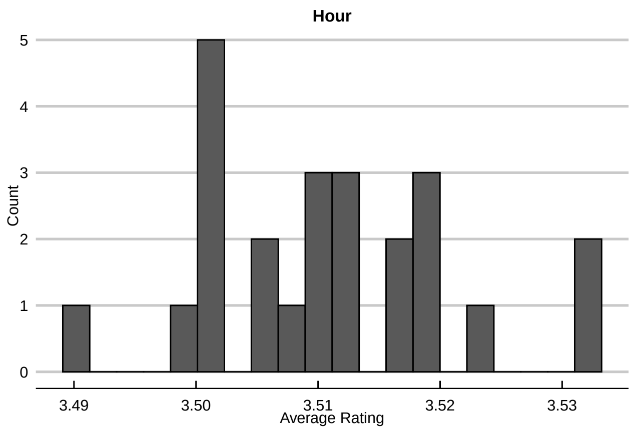












From visually inspecting the plots we can see that the variability of Date is largest and since all the other variable except for Hour are included we have decided to include Date in the next model and then we will include Hour as the last unincluded variable that we have created.

Regularized Movie + User + Genre + Date Effect

We build on the Regularized Movie + User + Genre Effect model, but here also include the Date Effect, which we also regularize to penalize the dates that have lesser numbers of movie ratings. Date consist of Year, Month and Month Day information.

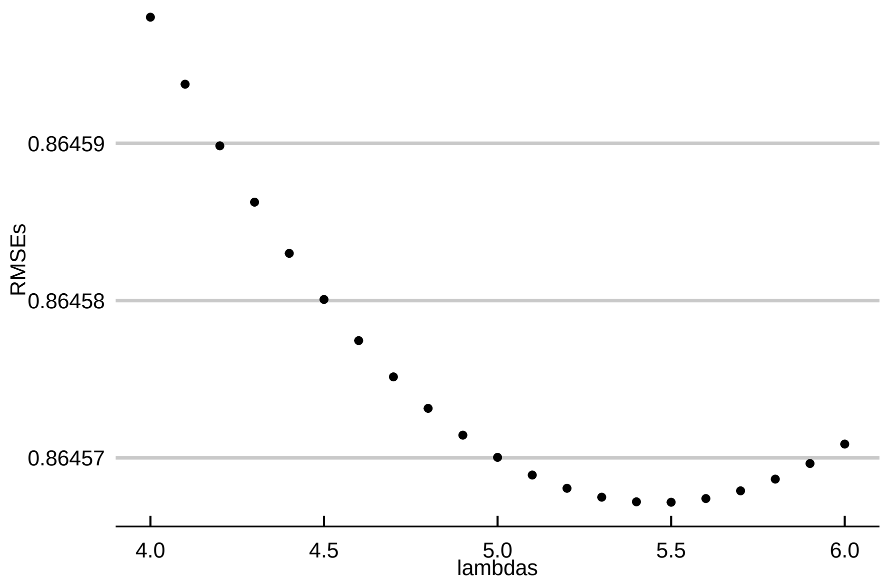
```
lambdas <- seq(4, 6, 0.1)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
  b_d <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    group_by(dates) %>%
    summarize(b_d = sum(rating - b_i - b_u - b_g - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_d, by = "dates") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_d) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
lamda_reg_mugd <- lambdas[which.min(rmses)]
```

Once done we select the lambda that optimized the model on the test set and we can display the results:

Table 12: Model Results

method	LAMBDA	RMSE_train
Movie Effect		0.94452
User Effects		0.97917
Movie + User Effects		0.88592
Regularized Movie Effect	1.5	0.94449
Regularized User Effect	5.5	0.97866
Regularized Movie + User Effect Model	5.1	0.86549
Regularized Movie + User + Genre Effect	5	0.86517
Regularized Movie + User + Genre + Date Effect	5.5	0.86457

We can plot the rmses vs. the lambdas to see that *lambda = 5.5* optimizes the model.



We have received a sufficient result in RMSE now, but lets see if we can do even better by including Hour variable in the mix.

Regularized Movie + User + Genre + Date + Hour Effect

We build on the Regularized Movie + User + Genre + Date Effect model, but here also include the Hour Effect, which we also regularize to penalize the dates that have lesser numbers of movie ratings. Before we start we have to add the Hour variable into training and test sets as previously during the data exploration we have done that on a secondary set.

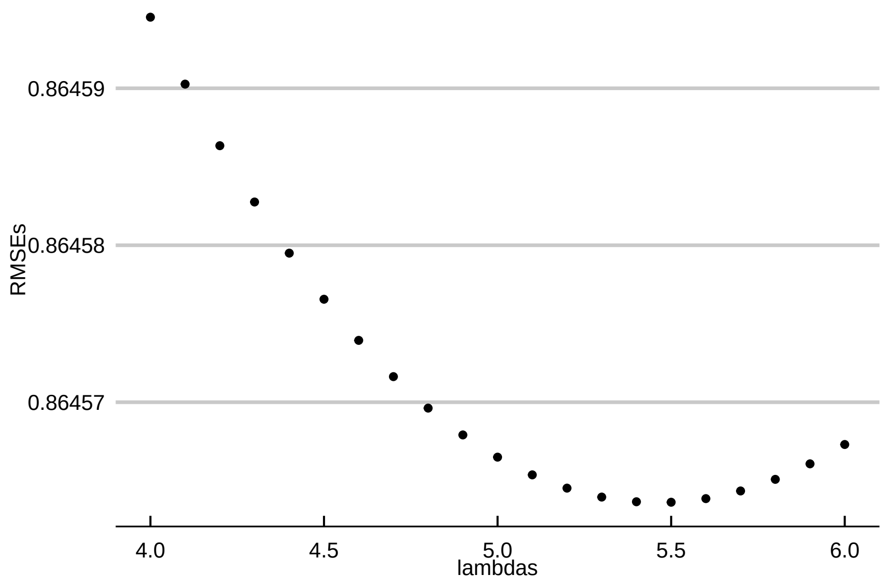
```
lambdas <- seq(4, 6, 0.1)
train_set<- train_set %>% mutate(hour=hour(as.POSIXct(train_set$timestamp,
                                                       origin="1970-01-01")))
test_set<- test_set %>% mutate(hour=hour(as.POSIXct(test_set$timestamp,
                                                     origin="1970-01-01")))
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
  b_d <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    group_by(dates) %>%
    summarize(b_d = sum(rating - b_i - b_u - b_g - mu)/(n()+l))
  b_h <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_d, by = "dates") %>%
    group_by(hour) %>%
    summarize(b_h = sum(rating - b_i - b_u - b_g - b_d - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_d, by = "dates") %>%
    left_join(b_h, by = "hour") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_d + b_h) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
lamda_reg_mugdh <- lambdas[which.min(rmses)]
```

Once done we select the lambda that optimized the model on the test set and we can display the results:

Table 13: Model Results

method	LAMBDA	RMSE_train
Movie Effect		0.94452
User Effects		0.97917
Movie + User Effects		0.88592
Regularized Movie Effect	1.5	0.94449
Regularized User Effect	5.5	0.97866
Regularized Movie + User Effect Model	5.1	0.86549
Regularized Movie + User + Genre Effect	5	0.86517
Regularized Movie + User + Genre + Date Effect	5.5	0.86457
Regularized Movie + User + Genre + Date + Hour Effect	5.5	0.86456

We can plot the rmses vs. the lambdas to see that *lambda = 5.5* optimizes the model.



We have not received any meaningful improvement and therefore we will consider the Regularizes Movie + User + Genre + Date Effect Model to be the final model we will use.

RESULTS

Training the Regularized Movie + User + Genre + Date Effect Model

We will be retraining the model on the entire edx set as opposed to the training set - this will allow us to create a further enhancement of the accuracy.

As we have previously included the Date variable only in the training and test sets we need to add it to the edx set as well.

```r
edx<- edx %>% mutate(dates=(as.Date(as.POSIXct(
  edx$timestamp, origin="1970-01-01"))))
```

We also need to select the lambda we are going to be using, as we are not going to be doing any further tuning we will select the optimized lambda from the training set.

```r
l<-lamda_reg_mugd
```

And now we can train our model on the edx set.

```r
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))
b_g <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
b_d <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  group_by(dates) %>%
  summarize(b_d = sum(rating - b_i - b_u - b_g - mu)/(n()+l))
```

Predicting on validation set and RESULT

Now we have a trained model and we can use it to predict the ratings on the validation set. NOTE: this is the first time the validation set has been used for any modeling activity in this project.

First we have to add the Date variable also to the Validation set:

```
validation<- validation %>% mutate(dates=(as.Date(as.POSIXct(
  validation$timestamp, origin="1970-01-01"))))
```

Next we create the predictions and calculate the final loss function (RMSE):

```
predicted_ratings <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_d, by = "dates") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_d) %>%
  .$pred
final_RMSE<- RMSE(predicted_ratings, validation$rating)
```

AND THE FINAL RESULT RECEIVED IS:

Table 14: FINAL MODEL RESULTS

method	LAMBDA	RMSE_train	**RMSE_final**
Movie Effect		0.94452	
User Effects		0.97917	
Movie + User Effects		0.88592	
Regularized Movie Effect	1.5	0.94449	
Regularized User Effect	5.5	0.97866	
Regularized Movie + User Effect Model	5.1	0.86549	
Regularized Movie + User + Genre Effect	5	0.86517	
Regularized Movie + User + Genre + Date Effect	5.5	0.86457	**0.86388**
Regularized Movie + User + Genre + Date + Hour Effect	5.5	0.86456	

CONCLUSION

The final model is a substantial improvement on a plain average prediction as well as non-regularized models. It is build of regularized effect of MOVIE, USER, GENRE and DATE.

Through basic and more thorough data exploration e have applied the principles learned in the classes of Data Science courses. To built a model that will include the effects of Genre and Date to further enhance the predictions. We have eliminated using also the Hour effect as it did not lead to any further enhancement of the model accuracy.

We have retrained the selected model on the whole edx set and then applied it to the validation set for final accuracy calculation.

THE FINAL RESULT OF MY REGULARIZED MOVIE + USER + GENRE + DATE EFFECT MODEL IS 0.86388.