200403_Capstone

Jason Schmidberger

03/04/2020

Movielens Project

Introduction

This document outlines the Movielens project that I am completing as part of my Data Science: Capstone module run by HarvardX through EdX. The module is being completed as the final part of the HarvardX Data Science Professional Certificate.

MovieLens is essentially a recommender system created by GroupLens Research in 1997. The full dataset contains over 26 million ratings of 45000 movies by 270000 users. A large community of online users utilise this package to develop and research recommender systems. It is the goal of this work to create a recommender system on a subset of the MovieLens dataset to show case the skills and techniques I have developed while undertaking the HarvardX Data Science course.

Initially, to construct the subset of the MovieLens dataset that will feature in this project, I followed clear instructions given in "Create Train and Validation Sets". Two dataframes are created that are to be carried forward into the data analysis and modelling part of the project. Dataframes "edx" which is the training dataset, and "validation" which is the smaller testing dataset.

The latter part of the report involves the development of a series of models/algorithms to be used to predict ratings based on predictors present in the final validation dataset. Monitoring of model quality is performed by performing an RMSE calculation using a cross validation portion of the edx training set. Only the final model will be assessed using an RMSE calculation comparing the model predictions to the validation testing dataset.

Methods/Analysis

In this section I avoid presenting prediction results as they will be presented in full in the next "Results" section. Here I focus on more general data analysis and presenting the code as my algorithm is developed.

Creation of training dataset "edx", and testing dataset "validation" was performed using code provided by HarvardX staff as part of instructions for this project. When run the code ionitially a 10 M subset of the full movielens dataset is used. This is split into 90% and 10% subsivisions for training and testing respectively. The code is not shown in the report but is imbedded in the Rmd file.

Preparation of cross validation subdivision of edx training dataset

As we can only use the validation testing dataset to assess the final model, I am setting up an aditional subdivision of the training dataset into edx_work (90%) that will be used for training the models, and edx_cv (10%) that will be used for cross validation (RMSE calculations).

```
test_index2 <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
edx_work <- edx[-test_index2,]
temp <- edx[test_index2,]
```

```
edx_cv <- temp %>%
  semi_join(edx_work, by = "movieId") %>%
  semi_join(edx_work, by = "userId")

# To ensure all movies and users present in edx_cv are also present in edx_work.
removed <- anti_join(temp, edx_cv)
```

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

```
edx_work <- rbind(edx_work, removed)
rm(test_index2, temp)
```

Development of Prediction algorithm

In this section I will begin with some basic analysis of the "edx" and "validation" datasets to get a better feel for them. Following that I will proceed through the development of a succession of prediction algorithms, each getting (ideally) better in terms of minimising an RMSE value.

Beginning with some basic analysis and data visualisation.

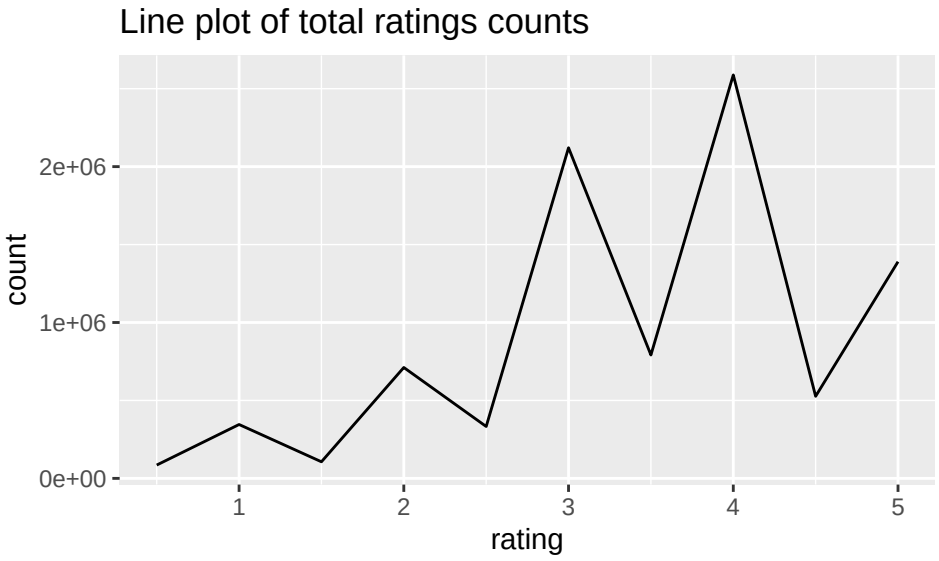
Top 10 movies with the highest number of ratings.

title	count
Pulp Fiction (1994)	31362
Forrest Gump (1994)	31079
Silence of the Lambs, The (1991)	30382
Jurassic Park (1993)	29360
Shawshank Redemption, The (1994)	28015
Braveheart (1995)	26212
Fugitive, The (1993)	25998
Terminator 2: Judgment Day (1991)	25984
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672
Apollo 13 (1995)	24284

To get a feeling for structure within the rating system it would be useful to rank the ratings values themselves (Most commonly used down to least).

rating	count
4.0	2588430
3.0	2121240
5.0	1390114
3.5	791624
2.0	711422
4.5	526736
1.0	345679
2.5	333010
1.5	106426
0.5	85374

Plotting this.



From this data, it is clear that 4 is the most common rating, and that half measures are far less common than non-decimal integers. An attampt was made to take advantage of this by rounding down the x.5 predicted ratings by their proportion in the dataset but it did not yield significant improvements to the RMSE value of that model.

Defining the RMSE algorithm used assess usefulness of prediction algorithms.

RMSE = sqrt(mean((true_ratings - predicted_ratings)^2))

Prediction algorithm #1: Just the average.

In what is the most basic model, as I guess ratings based on movie and user data, I assume they are all equal to the over all mean of all ratings in edx (mu_hat). See value below.

	mean(edx_work$rating)
mu_hat	3.512509

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

Prediction algorithm #2: Movie effect model

There are certainly situations when some movies have a tendency to perform particularly well or particularly poorly in ratings. Here is an attempt to adjust the average rating value by a specific movie specific adjustment (b_i). This b_i value for any given movie is equal to the mean of all the residuals obtained by subtracting each rating from the overall average rating (mu).

```
mu <- mean(edx_work$rating)
movie_avgs <- edx_work %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))
head(movie_avgs, 10) %>% knitr::kable()
```

movieId	b_i
1	0.4153068
2	-0.3033248
3	-0.3590946
4	-0.6313877
5	-0.4434507
6	0.3028756
7	-0.1540214
8	-0.3677188
9	-0.5260159
10	-0.0882810

Above you can see the b_i values for movieId's 1 through 10.


```
# Calculating the new predictions given the b_i values for each row in edx_cv dataset and adding it to
# mu.
predicted_ratings_2 <- mu + edx_cv %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

model_2_rmse <- RMSE(predicted_ratings_2, edx_cv$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Movie Effect Model",
                                     RMSE = model_2_rmse))
```

To get the new predictions the b_i value for each movie is added to the average rating (mu).

Prediction algorithm #3: Movie and user effect model

Obviously users can also vary in how they rate movies. Some may be very generous, while others can be quite difficult to impress. Consequently applying a b_u modifier to the model will likely provide an advantage. This b_u value is the mean of mu - the actual ratings for each user of interest.

So applying both the b_i value calculated above, and this new b_u value (collectly called "User Effect Model") reduced the RMSE value much more to 0.8850 (see table below).

```
### Now I am looking at the influence of the user in context of the movie effect (i.e. cumulative effec
user_avgs <- edx_work %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu))
predicted_ratings_3 <- edx_cv %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings_3, edx_cv$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "User and Movie Effect Model",
                                     RMSE = model_3_rmse))
```

Regularisation

In a number of cases there is a tendency for large b_i or b_u values that are misrepresentative of the movies or users they represent. This happens when n is small, as in these cases extreme ratings (i.e. unusually low or high ratings) will have a bigger impact on the average for that user or movie. In order to correct for situations such as these, all b_i and b_u values can have a regularisation applied to them to effectively scale down excessive values when n is small.

Having a look at the top 10 movies with largest b_i residuals. These movies differ the most from the average movie rating of 3.512, in the sense that they rate better.

title	b_i
Hellhounds on My Trail (1999)	1.487491
Satan's Tango (Sátántangó) (1994)	1.487491
Shadows of Forgotten Ancestors (1964)	1.487491
Fighting Elegy (Kenka erejii) (1966)	1.487491
Sun Alley (Sonnenallee) (1999)	1.487491
Bullfighter and the Lady (1951)	1.487491
Blue Light, The (Das Blaue Licht) (1932)	1.487491
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237491
I'm Starting From Three (Ricomincio da Tre) (1981)	1.237491
Human Condition II, The (Ningen no joken II) (1959)	1.237491

Now looking at the 10 movies with lowest residuals. That is, movies that are much lower than the average rating.

title	b_i
Besotted (2001)	-3.012509
Hi-Line, The (1999)	-3.012509

title	b_i
Accused (Anklaget) (2005)	-3.012509
Confessions of a Superhero (2007)	-3.012509
War of the Worlds 2: The Next Wave (2008)	-3.012509
SuperBabies: Baby Geniuses 2 (2004)	-2.679176
Disaster Movie (2008)	-2.641541
Hip Hop Witch, Da (2000)	-2.637509
From Justin to Kelly (2003)	-2.616905
Criminals (1996)	-2.512509

You can see from the two tables above that both top 10 and bottom 10 include quite obscure movies.

If we take another look at the top 10 list, but include the number of ratings for each.

```
## Joining, by = "movieId"
```

title	b_i	n
Hellhounds on My Trail (1999)	1.487491	1
Satan's Tango (Sátántangó) (1994)	1.487491	1
Shadows of Forgotten Ancestors (1964)	1.487491	1
Fighting Elegy (Kenka erejii) (1966)	1.487491	1
Sun Alley (Sonnenallee) (1999)	1.487491	1
Bullfighter and the Lady (1951)	1.487491	1
Blue Light, The (Das Blaue Licht) (1932)	1.487491	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237491	4
I'm Starting From Three (Ricomincio da Tre) (1981)	1.237491	2
Human Condition II, The (Ningen no joken II) (1959)	1.237491	4

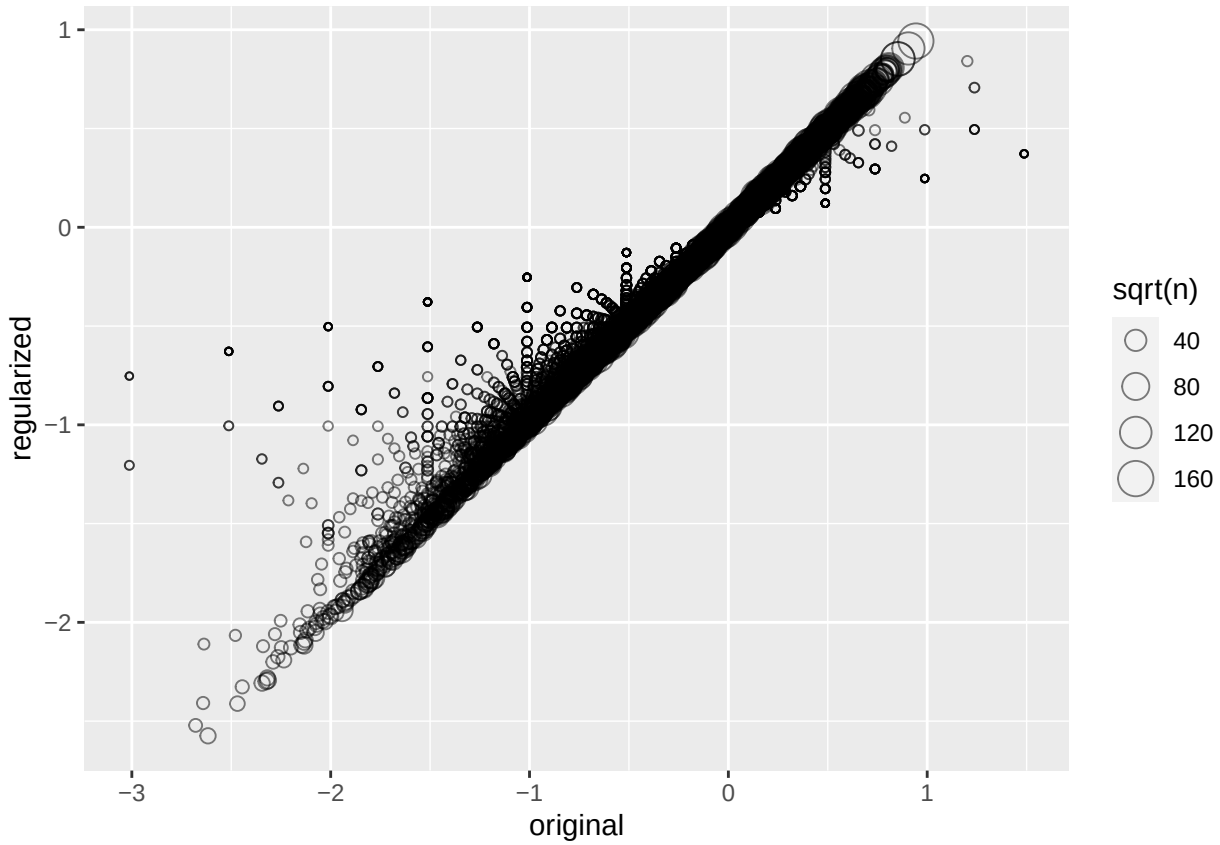
Application of the regularising term "lambda".

To begin with, lambda will be set at a value of 3.

movieId	b_i	n_i
1	0.4152488	21459
2	-0.3032305	9647
3	-0.3589243	6326
4	-0.6300462	1409
5	-0.4432197	5756
6	0.3027939	11115
7	-0.1539507	6533
8	-0.3662320	739
9	-0.5252420	2036
10	-0.0882617	13699

The new regularised b_i values and their corresponding n_i values are displayed.

Below is a figure that illustrates the difference in b_i values from the original movie_avgs vs the modified movie_reg_avgs. The smaller the value of n associated with with a b_i, the smaller the circle.



Most of the data points are in a roughly linear slope. Values with smaller circles (small n) have a tendency to have values closer to zero in the regularised data. This is due to the lambda in the equation having more significance when n is a low value.

If we now take a look look at top 10 residuals for movies using these regularised movie estimates, now we are seeing more sensible movies start to appear, with understandably lower b_i values.

```
## Joining, by = "movieId"
```

title	b_i	n
Shawshank Redemption, The (1994)	0.9430405	25204
Godfather, The (1972)	0.9053536	16055
Usual Suspects, The (1995)	0.8521278	19570
Schindler's List (1993)	0.8520946	20854
More (1998)	0.8412437	7
Casablanca (1942)	0.8075522	10078
Rear Window (1954)	0.8041232	7130
Seven Samurai (Shichinin no samurai) (1954)	0.8019221	4677
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	0.8013964	2603
Double Indemnity (1944)	0.7992293	1917

Prediction algorithm #4: Regularised movie effect

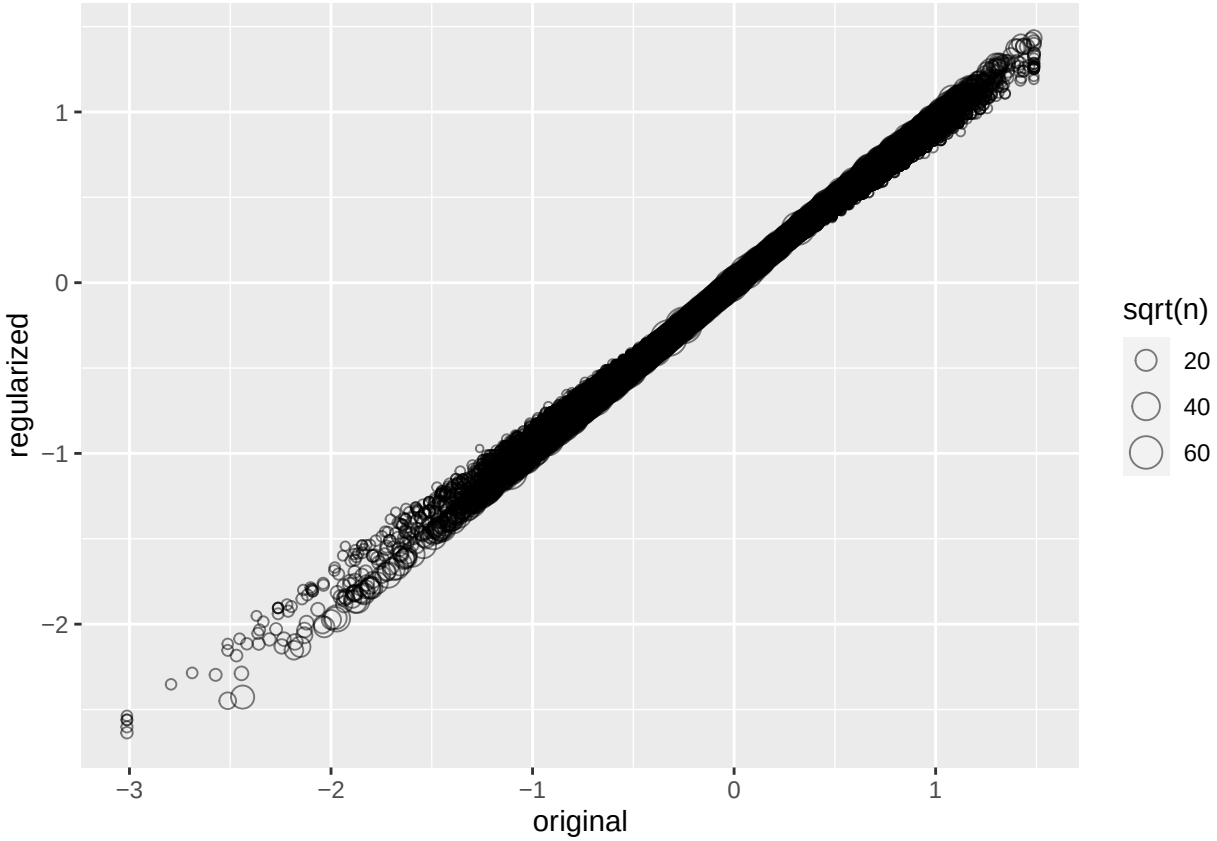
Applying the regularised b_i values to the prediction.

```
# Perform prediction on regularised movie b_i.
predicted_ratings_4 <- edx_cv %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
```

```
  mutate(pred = mu + b_i) %>%
  .$pred

model_4_rmse <- RMSE(predicted_ratings_4, edx_cv$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Regularised movie effect",
                                     RMSE = model_4_rmse))
```

The RMSE for "Regularised movie effect" is only slightly better than the unregularised movie effect (see results section).

Now to I need to apply the same calculation to regularise the user effects. Below is plot of original b_u versus regularised b_u. This plot varies less than the plot equivalent plot for b_i seen above.



Prediction algorithm #5: Regularised user effect

Applying this regularised b_u to the model.

```
predicted_ratings_5 <- edx_cv %>%
  left_join(user_reg_avgs, by = "userId") %>%
  mutate(pred = mu + b_u) %>%
  .$pred

model_5_rmse <- RMSE(predicted_ratings_5, edx_cv$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Regularised user effect",
                                     RMSE = model_5_rmse))
```

Interestingly this regularised b_u is worse than the origial b_u values in the prediction model.

Prediction algorithm #6: Reg movie and user effect

Applying both regularised values at the same time.

```
# Apply regularised user b to whole prediction calculation.
predicted_ratings_6 <- edx_cv %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  left_join(user_reg_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_6_rmse <- RMSE(predicted_ratings_6, edx_cv$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Reg movie and user effect",
                                     RMSE = model_6_rmse))
```

Together these provide an over all improvement compared to their unregularised equivalents (see results section).

Parameterisation of Lambda

Lambda is a parameterisable variable. It is likely that I could optimise is to return lower RMSE values.

I will set lambda to be values between 0 and 10, with increments of 0.25.

```
lambda <- seq(0, 10, 0.25)
```

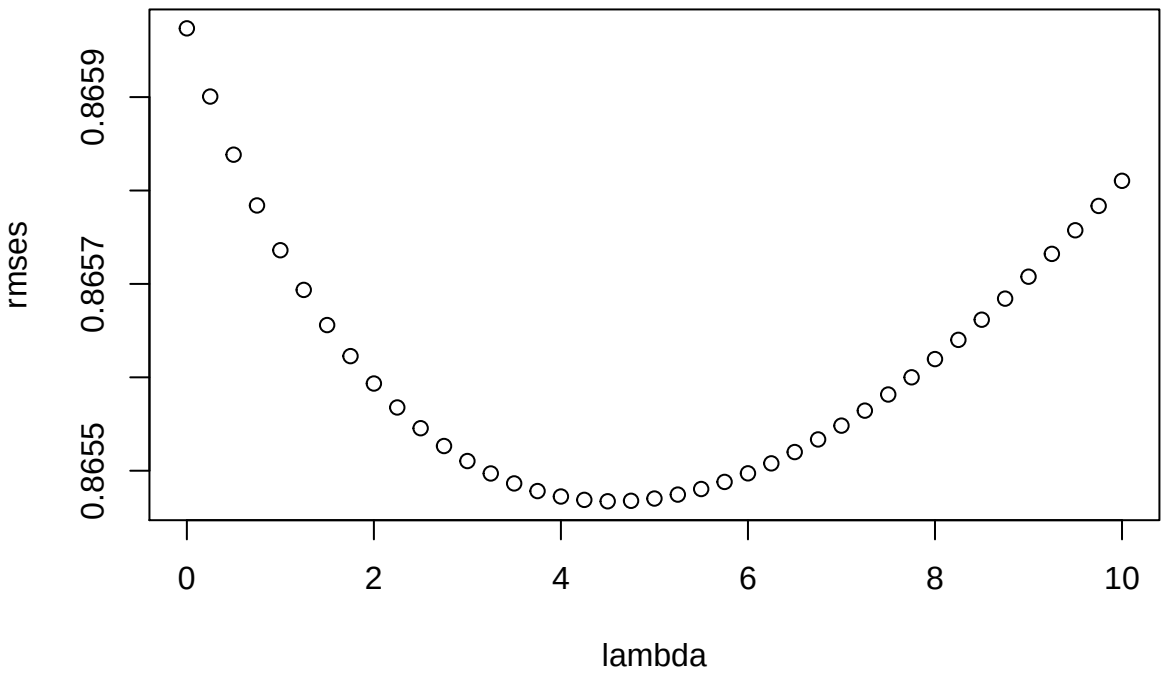
I will tune lambda against the edx_cv subset of the edx training dataset. I cannot use the testing (validation) dataset until I make the final RMSE calculation.

Now to run the tuning procedure.

```
# Running the refinement of best lambda value.
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
rmses <- sapply(lambda, function(l){
  mu <- mean(edx_work$rating)
  b_i <- edx_work %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/(n()+l))
  b_u <- edx_work %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings<- edx_cv %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, edx_cv$rating))
})
plot(lambda, rmses, main = "Tuning Lambda")
```




```
## [1] "The optimal Lambda"
```

```
## [1] 4.5
```

The lambda value that returns the lowest RMSE is 4.5

Prediction algorithm #7: Reg mov & usr with 4.5 l

Feeding this value formally into the prediction algorithm.

```
b_i <- edx_work %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu)/(n()+4.5))
b_u <- edx_work %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - b_i - mu)/(n()+4.5))
predicted_ratings_7 <- edx_cv %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
model_7_rmse <- RMSE(predicted_ratings_7, edx_cv$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Reg mov & usr with 4.5 l",
                                     RMSE = model_7_rmse))
rm(edx, predicted_ratings_2, predicted_ratings_3, predicted_ratings_4, predicted_ratings_5, predicted_ra
```

Principle component analysis

In an effort to improve on the regularisation of movie (b_i) and user (b_u) residuals I decided to implement a principle component analysis to detect any relationships that exist in the data between types of movies and

the users rating them.

Through trial and error I have determined that in order to perform a PCA, I would have to take a further subset of the "edx" training dataset. Performing a calculation on the full "edx" is computationally too expensive to be practical using this technique. As such I have limited the dataset to include instances of movies with >1000 ratings, and users that gave >500 ratings ("edx_small").

```
# Focusing in on the most rated movies (>1000 ratings), and most prolific users (>500 ratings).
edx_small <- edx_work %>%
  group_by(movieId) %>%
  filter(n() >= 1000) %>% ungroup() %>%
  group_by(userId) %>%
  filter(n() >= 500) %>% ungroup()
```

Next I would need to convert this dataset into a matrix with columns representing movieId numbers, and rows representing userId numbers. Where they intersect are the ratings associated with each user/movie combination.

```
y <- edx_small %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()
y[1:10, 1:5]
```

```
##       userId   1   2  3  4
##  [1,]    143 2.0  NA  3 NA
##  [2,]    182 4.0 3.0  3  4
##  [3,]    215  NA 4.0 NA NA
##  [4,]    276 3.5 1.5 NA NA
##  [5,]    289 4.0  NA NA NA
##  [6,]    426 3.5 1.5 NA NA
##  [7,]    533 5.0 4.0 NA NA
##  [8,]    543 4.0 4.0 NA NA
##  [9,]    585  NA  NA NA  2
## [10,]    657 3.5  NA  2 NA
```

```
rm(edx_small, edx_work)
```

Taking now the first column and using the userId values as rownames for the matrix and then removing them from the matrix to ensure only ratings values are included in the cells.

```
##       1   2  3  4  5
## 143 2.0  NA  3 NA  4
## 182 4.0 3.0  3  4  4
## 215  NA 4.0 NA NA NA
## 276 3.5 1.5 NA NA NA
## 289 4.0  NA NA NA NA
## 426 3.5 1.5 NA NA NA
## 533 5.0 4.0 NA NA NA
## 543 4.0 4.0 NA NA NA
## 585  NA  NA NA  2 NA
## 657 3.5  NA  2 NA NA
```

A temporary file (tmp) is created here for use later in a figure (see below).

The matrix is then prepared for the PCA by converting the ratings to difference residuals, subtracting both the user average and movie averages successively.

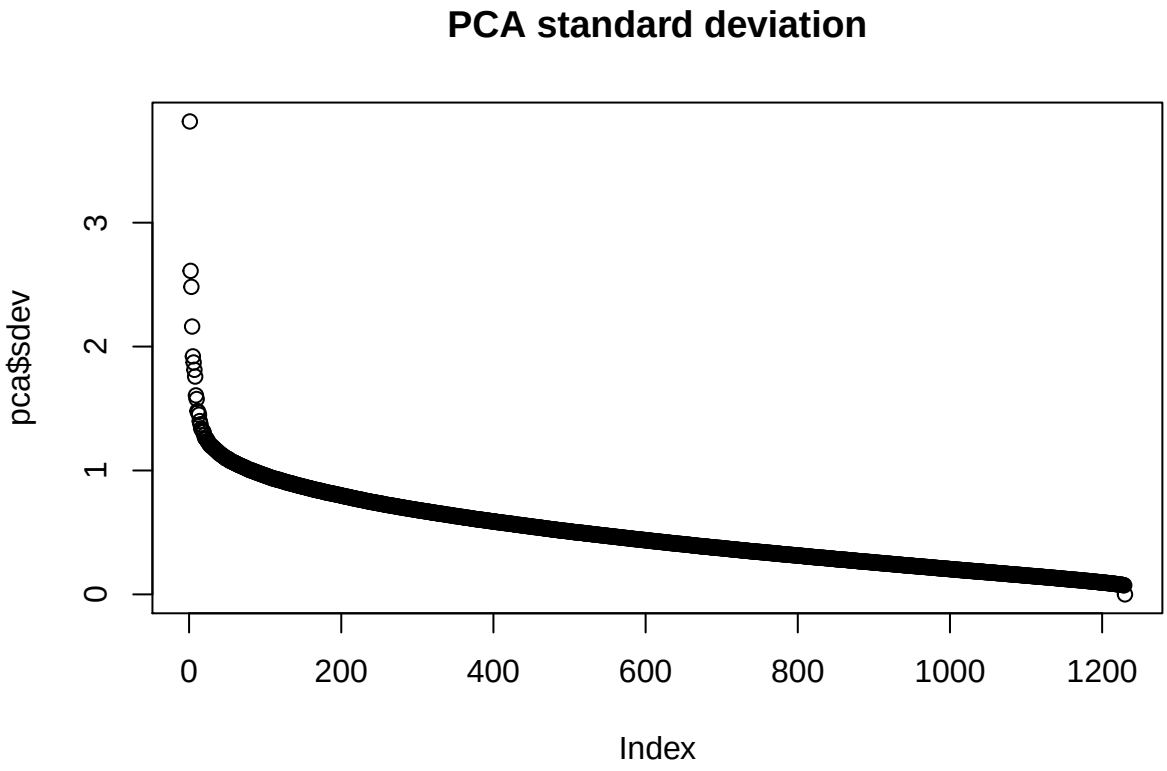
```r
### Preparing the matrix for PCA.
y <- sweep(y, 1, rowMeans(y, na.rm = TRUE))
y <- sweep(y, 2, colMeans(y, na.rm = TRUE))

y[is.na(y)] <- 0
y <- sweep(y, 1, rowMeans(y))
```

The data is zeroed by means of determining the differentials. There is no need for scaling as all descriptors stem from the same rating scale.

executing the pca.

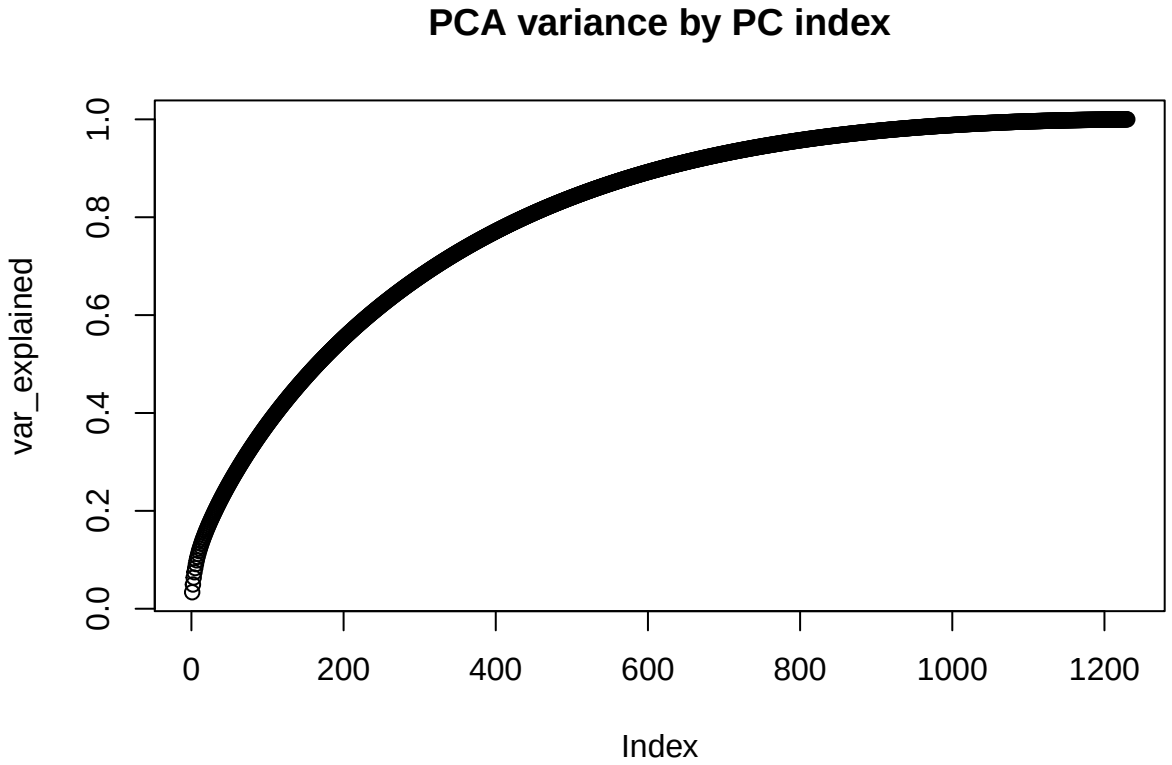
```r
pca <- prcomp(y, center = F, scale. = F, retx = T)
```

Visualising the PCA. Plot of standard deviation.



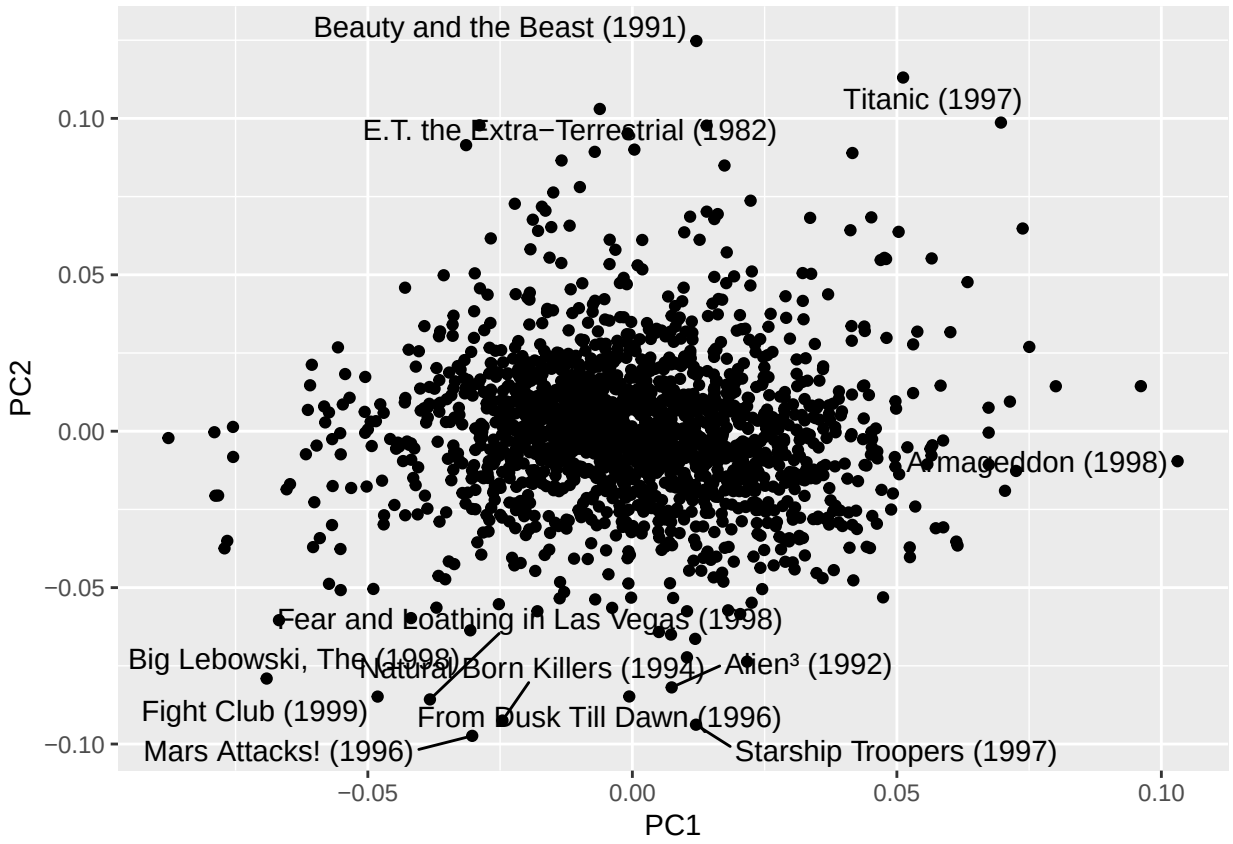
Plot of variance.

```r
var_explained <- cumsum(pca$sdev^2/sum(pca$sdev^2))
plot(var_explained, main = "PCA variance by PC index")
```



Certainly most of the variation is explained by 500 principal components. It is impractical for me to apply all of these using the method I implement below. I will begin by applying a few PC's at a time.

Some more data visualisation. Dealing with just the principle components themselves (i.e. pca$rotation output), and then looking at the plot of PC1 vs PC2 to get a feel for the sort of relationships that exist between movies. (This is where the tmp file is used)



Here is a list of the bottom 10 movies with lowest PC1 values.

name	PC1
2001: A Space Odyssey (1968)	-0.0876942
Fargo (1996)	-0.0790072
Taxi Driver (1976)	-0.0788093
Rushmore (1998)	-0.0783005
Clockwork Orange, A (1971)	-0.0770914
Royal Tenenbaums, The (2001)	-0.0765708
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)	-0.0754940
Being John Malkovich (1999)	-0.0754605
Big Lebowski, The (1998)	-0.0691255
Pulp Fiction (1994)	-0.0667902

Here is a list of the top 10 movies with highest PC1 values.

name	PC1
Armageddon (1998)	0.1030721
Independence Day (a.k.a. ID4) (1996)	0.0961242
Twister (1996)	0.0800476
Top Gun (1986)	0.0750382
Ghost (1990)	0.0737872
Batman & Robin (1997)	0.0725400
Patriot, The (2000)	0.0713729
Lethal Weapon 4 (1998)	0.0704403
Pretty Woman (1990)	0.0696758

name	PC1
Patch Adams (1998)	0.0673817

SETTING UP THE EDX_CV DATA FRAME TO BE COMPATIBLE WITH PCA.

Now I need the actual ratings from the edx_cv data set in a 68052 by 9728 matrix. The reason these are the necessary dimensions is because this is are the numbers of unique users, and movies (respectively) in the testing ("edx_cv") dataframe.

```
val_ratings.m <- edx_cv %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()
```

Taking the first column values as row names, then deleting it. Column names are taken as the movie titles.

```
rownames(val_ratings.m) <- val_ratings.m[,1]
val_ratings.m <- val_ratings.m[,-1]
colnames(val_ratings.m) <- with(movie_titles, title[match(colnames(val_ratings.m), movieId)])
val_ratings.m[1:10, 5:7]
```

```
##    Father of the Bride Part II (1995) Heat (1995) Sabrina (1995)
## 1                                  NA          NA             NA
## 2                                  NA          NA             NA
## 3                                  NA          NA             NA
## 4                                  NA          NA             NA
## 5                                  NA          NA             NA
## 6                                  NA          NA             NA
## 7                                  NA          NA             NA
## 8                                  NA          NA             NA
## 9                                  NA          NA             NA
## 10                                 NA          NA              3
```

You will see from the small snippet above, that this is a sparse matrix. The first actual rating is in row 10, columm 7.

For the matrix operations that I will apply later, it is important that the dimensions of my PCA data matches the edx_cv matrix above (i.e. dimensions 68052 by 9728).

I need to take the x and rotation matrices from pca and expand them to match the dimensions of the edx_cv matrix. That is unique userId (68052) by unique movieId (9728). These expanded matrices will be sparse matrices.

To start with, I need the lists of all userId's and movieId's in the edx_cv dataframe to use as a reference.

First userId.

```
unique_usr_val <- as.matrix(unique(edx_cv$userId))
colnames(unique_usr_val) <- c("userId")
```

Now MovieId.

```
unique_mov_val <- as.matrix(unique(edx_cv$movieId))
colnames(unique_mov_val) <- c("movieId")
```

Creating the "User effect" sparse matrix. All NA's of this sparse matrix are converted to 0.

```
pca_x <- pca$x %>% as.data.frame() %>%
  tibble::rownames_to_column(., "userId") %>%
```

```
  merge(unique_usr_val, ., by = "userId", all = TRUE)
pca_x <- as.matrix(pca_x[,-1])
pca_x[is.na(pca_x)] <- 0
```

Creating the "Principal component" sparse matrix. All NA's of this sparse matrix are converted to 0.

```
pca_rotation <- pca$rotation %>% as.data.frame() %>%
  tibble::rownames_to_column(., "movieId") %>%
  merge(unique_mov_val, ., by = "movieId", all = TRUE)
pca_rotation <- as.matrix(pca_rotation[,-1])
pca_rotation[is.na(pca_rotation)] <- 0
```

Now to build a large matrix of predictions from where I left off after "Reg mov and usr with 4.5". This will have the dimensions that are consistent with unique userId/movidId values.

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

Below, y3 is the matrix of predictions made above, before I apply any of the PCA corrections. The dimensions of this matrix match the edx_cv matrix (68052 by 9728).

```
y3 <- predictions %>%
  select(userId, movieId, pred) %>%
  spread(movieId, pred) %>%
  as.matrix()
rownames(y3) <- y3[,1]
y3 <- y3[,-1]
```

To apply the PCA outcomes to my prediction algorithm, it will take the form of; pred = mu + b_i + b_u + p1q1 + p2q2 + ... pnqn,

where p and q are vectors containing the a single column of user effect (pca$x$)$and principle component(pca$rotation) from the PCA, respectively. Each pq pair represents a successive principle component element to be applied to the prediction algorithm. p1q1 is equivalent to the effect ofPC1, and so on. Multiplying the vectors together will generate a matrix of dimensions 68052 by 9728. This can be added to the previous prediction calculations to effect the changes introduced by each successive principle component.

Performing the new prediction using PC1. Remembering y3 is the matrix of predictions from the last step of regularisation and application of lambda correction for both user and movie effect (i.e. "Reg mov & usr with 4.75 l").

Defining my p and q vectors Vectors p are called user effects. Vectors q are called prinipal components. NOTE... both p1 and q1 are converted to matrices, with p1 being 68052 X 1, and q1 1 X 9728

```
p1 <- as.matrix(pca_x[,1])
q1 <- matrix(pca_rotation[,1], nrow = 1, byrow = T)
p2 <- as.matrix(pca_x[,2])
q2 <- matrix(pca_rotation[,2], nrow = 1, byrow = T)
p3 <- as.matrix(pca_x[,3])
q3 <- matrix(pca_rotation[,3], nrow = 1, byrow = T)
p4 <- as.matrix(pca_x[,4])
q4 <- matrix(pca_rotation[,4], nrow = 1, byrow = T)
p5 <- as.matrix(pca_x[,5])
q5 <- matrix(pca_rotation[,5], nrow = 1, byrow = T)
p6 <- as.matrix(pca_x[,6])
q6 <- matrix(pca_rotation[,6], nrow = 1, byrow = T)
```

```
p7 <- as.matrix(pca_x[,7])
q7 <- matrix(pca_rotation[,7], nrow = 1, byrow = T)
p8 <- as.matrix(pca_x[,8])
q8 <- matrix(pca_rotation[,8], nrow = 1, byrow = T)
p9 <- as.matrix(pca_x[,9])
q9 <- matrix(pca_rotation[,9], nrow = 1, byrow = T)
p10 <- as.matrix(pca_x[,10])
q10 <- matrix(pca_rotation[,10], nrow = 1, byrow = T)
```

It is important to convert these vectors into actual matrices of dimentions 'x X 1' and '1 X y' so they can be multiplied together as part of the prediction algorithm. This will generates matrices of dimensions x by y.

Prediction algorithm #9: Reg plus PC1 to PC10

Adding the first 10 principle components to the previous prediction model (i.e. "REG MOV & USR WITH 4.5 L").

```
new_pred_10 <- y3 + (p1%*%q1) + (p2%*%q2) + (p3%*%q3) + (p4%*%q4) + (p5%*%q5) + (p6%*%q6) +
  (p7%*%q7) + (p8%*%q8) + (p9%*%q9) + (p10%*%q10)

rmse_PC1_to_10 <- sqrt(mean((val_ratings.m - new_pred_10)^2, na.rm = TRUE))

rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Reg plus PC1 to PC10",
                                     RMSE = rmse_PC1_to_10))
```

I can likely push this further throught the addition of more PC's, but the computation requirements necessary are beginning to outweight the gains in RMSE values. I will stop here at PC's 1 to 10, and a final RMSE value of 0.861.

Results

Above I have run through the methodology that resulted in my final prediction algorithm for the 10M movielens database. I will now present the outcomes of each stage of my prediction algorithm and discuss each stage in some detail.

"JUST THE AVERAGE" - RMSE value = 1.061

Obviously this is not a great value. Being on average more than 1 unit away from the true rating in my prediction system when the whole raring system only ranges from 0.5 to 5, is not an impressive prediction system. Clearly some improvement on just taking the average rating is possible. This should just be considered a benchmark of sorts.

"MOVIE EFFECT MODEL" - RMSE value = 0.944

Now we have some improvement. Dropping just under an RMSE value of 1 but taking into account how the aveage ratings for each separate movie differ from the overall average. Obviously some movies are on average rated higher than others. Some lower too. Applying these residuals to the average improves the model RMSE by just over 11% to 0.944.

"USER AND MOVIE EFFECT MODEL" - RMSE value = 0.886

I did not look at the influence of the user effect alone, but rather jumped to the combination of movie effect plus the user effect together. Taking these two together made a significant improvement of the model RMSE to 0.885. Now this is looking decent.

"REGULARISED MOVIE EFFECT" - RMSE value = 0.944

Taking a step backward an only looking at the movie effect alone, but after it has been regularised to take in to account cases where residuals are inflated by low n values. In the first case a somewhat random value of lambda (i.e. 3) is used and a negligible improvement in the model RMSE is attained. From 0.94391 for unregularised residuals, to 0.94385 for regularised residuals (lambda = 3).

"REGULARISED USER EFFECT" - RMSE value = 0.979

For what ever reason, this model performed worse than any other model yet, excluding the base level "Just the average" model. This may be explained by the randomly chosen lambda value of 3. Some optimisation may be required to make regularisation of the user effect useful.

"REG MOVIE AND USER EFFECT" - RMSE = 1.26

Combining the regularised movie and regularised user effects had a dramitically negative effect on the RMSE value of the model. It will be interesting to see if the lambda parameter could be optimised to return an even better RMSE.

"REG MOV & USR WITH 4.5 L" - RMSE = 0.865

Cross validation of the lambda value against the edx_cv subset returned a value of 4.5 which yielded a model RMSE of 0.865 which is quite impressive. This is a remarkable turn around as obviously the original assignment of lambda to 3 was non-ideal and refining this parameter had a tremendous effect on the model's effectiveness.

"REG PLUS PC1 to PC10" - RMSE = 0.861

Adding the first 10 PC's dropped the model RMSE down to 0.861 which is quite good for this recomendation system. To give some context to this, the Netflix prize that offered $1M to a successful team, if that team to drop the benchmark RMSE of the time by 10% to 0.8572.

Below is a table outlining all the prediction models and their coresponding RMSE values when evaluated against edx_cv.

method	RMSE
Just the average	1.0611350
Movie Effect Model	0.9441568
User and Movie Effect Model	0.8859483
Regularised movie effect	0.9441230
Regularised user effect	0.9791932
Reg movie and user effect	0.8843992
Reg mov & usr with 4.5 l	0.8654673
Reg plus PC1 to PC10	0.8605152

FINAL Prediction Algorithm

As a final model I am taking that last algorithm (Reg plus PC1 to PC10) which is a culmination of all preceding algorithms. A final assessment for the algorithm is to test it against the validation testing dataset that has been kept aside during the course of this work.

However as the last PCA section is based on matrix operations that must match dimensions perfectly, some modifications are necessary to test it against the validation dataset.

First the validation dataset has to be put into matrix form to make it compatible with the PCA work.

```
validation.m <- validation %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()
rownames(validation.m) <- validation.m[,1]
validation.m <- validation.m[,-1]
colnames(validation.m) <- with(movie_titles, title[match(colnames(validation.m), movieId)])
```

The dimensions of this validation matrix are 68534 by 9809. This is larger than the edx_cv matrix used previously which was 68052 by 9728. Hense the need for modification here.

Next unique lists of users and movies must be determined for validation dataset. First userId.

```
unique_usr_val.f <- as.matrix(unique(validation$userId))
colnames(unique_usr_val.f) <- c("userId")
```

Now MovieId.

```
unique_mov_val.f <- as.matrix(unique(validation$movieId))
colnames(unique_mov_val.f) <- c("movieId")
```

Creating the final model "User effect" sparse matrix. All NA's of this sparse matrix are converted to 0.

```
pca_x.f <- pca$x %>% as.data.frame() %>%
  tibble::rownames_to_column(., "userId") %>%
  merge(unique_usr_val.f, ., by = "userId", all = TRUE)
pca_x.f <- as.matrix(pca_x.f[,-1])
pca_x.f[is.na(pca_x.f)] <- 0
```

Creating the final model "Principal component" sparse matrix. All NA's of this sparse matrix are converted to 0.

```
pca_rotation.f <- pca$rotation %>% as.data.frame() %>%
  tibble::rownames_to_column(., "movieId") %>%
```

```
  merge(unique_mov_val.f, ., by = "movieId", all = TRUE)
pca_rotation.f <- as.matrix(pca_rotation.f[,-1])
pca_rotation.f[is.na(pca_rotation.f)] <- 0
```

Now to rebuild a large matrix of predictions based this time on the validation matrix, from where I left off after "Reg mov and usr with 4.5". This will have the dimensions that are consistent with validation dataset unique userId/movidId values.

```
predictions.f <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u)
```

Below, y3.f is the matrix of new predictions made above, before I apply any of the PCA corrections. The dimensions of this matrix match the validation matrix (68534 by 9809).

```
y3.f <- predictions.f %>%
  select(userId, movieId, pred) %>%
  spread(movieId, pred) %>%
  as.matrix()
rownames(y3.f) <- y3.f[,1]
y3.f <- y3.f[,-1]
```

I have to define the p and q vectors again as their lengths are linked to the dimensions of the original edx_cv unique users, and unique movies respectively. The dimantions of the validation dataset equivalents are different.

```
p1 <- as.matrix(pca_x.f[,1])
q1 <- matrix(pca_rotation.f[,1], nrow = 1, byrow = T)
p2 <- as.matrix(pca_x.f[,2])
q2 <- matrix(pca_rotation.f[,2], nrow = 1, byrow = T)
p3 <- as.matrix(pca_x.f[,3])
q3 <- matrix(pca_rotation.f[,3], nrow = 1, byrow = T)
p4 <- as.matrix(pca_x.f[,4])
q4 <- matrix(pca_rotation.f[,4], nrow = 1, byrow = T)
p5 <- as.matrix(pca_x.f[,5])
q5 <- matrix(pca_rotation.f[,5], nrow = 1, byrow = T)
p6 <- as.matrix(pca_x.f[,6])
q6 <- matrix(pca_rotation.f[,6], nrow = 1, byrow = T)
p7 <- as.matrix(pca_x.f[,7])
q7 <- matrix(pca_rotation.f[,7], nrow = 1, byrow = T)
p8 <- as.matrix(pca_x.f[,8])
q8 <- matrix(pca_rotation.f[,8], nrow = 1, byrow = T)
p9 <- as.matrix(pca_x.f[,9])
q9 <- matrix(pca_rotation.f[,9], nrow = 1, byrow = T)
p10 <- as.matrix(pca_x.f[,10])
q10 <- matrix(pca_rotation.f[,10], nrow = 1, byrow = T)
```

Final Prediction algorithm

Adding the first 10 principle components to the previous prediction model (i.e. "REG MOV & USR WITH 4.5 L").

```
new_pred_10.f <- y3.f + (p1%*%q1) + (p2%*%q2) + (p3%*%q3) + (p4%*%q4) + (p5%*%q5) + (p6%*%q6) +
  (p7%*%q7) + (p8%*%q8) + (p9%*%q9) + (p10%*%q10)
```

```r
rmse_final <- sqrt(mean((validation.m - new_pred_10.f)^2, na.rm = TRUE))
print("Final Algorithm RMSE")
```

```
## [1] "Final Algorithm RMSE"
```

```r
rmse_final
```

```
## [1] 0.8602696
```

The RMSE value for the final prediction algorithm is 0.860

Conclusion

This body of work produces a series of recommender systems that serve to predict the ratings of a dataset of user/movie ratings. Starting with a readily available 10M MovieLens dataset, it was broken up into a training dataset ("edx") and a testing dataset ("validation") consisting of 90% and 10% of the original data respectively. Models were trained against the training dataset and had the RMSE's of their predictions determined by comparing them to the testing dataset.

The development of the prediction algorithm picked up from instructions given in section 6 of the HarvardX Machine Learning module, and applied them to this larger MovieLens dataset. RMSE values consistently dropped in common to the section 6 worked example. It is worth noting that a different optimal lambda value was found in this work, and the RMSE value returned by the application of this lambda value already gave an impressive 0.8648. It should also be noted that cross validation of the lambda parameter against the sub-divided training dataset was new to this investigation.

Performing the PCA also followed base instructions given in section 6 of the HarvardX course. However, application of the results of this PCA analysis is new to this work. Careful construction of a series of matrices that enabled the base prediction equation (pred = mu + b_i + b_u + p1q1 + p2q2 + ... pnqn). The resulting application of the top 10 principle components gave incremental improvements to the model RMSE, culminating in an impresive final result for the top 10 PC's and an RMSE of 0.8579. There is no doubt that application of more PC's will drop the RMSE further, but to a deminishing extent, and due to the high computational requirement, the decision has been made to stop at 10. Still even with the best 10 principle components, the prediction algorithm performs well.