Movielens Capstone Project

Nick Bova

19 August, 2020

1 Introduction

This project has been prepared as part of the HarvardX Data Science Series[1] Capstone Course. This project is an extension of the Movielens analysis that was introduced in the Machine Learning Course. During that course the base model was introduced and run against a smaller version of the data set containing 1 million records. In this project we will use a data set that contains 10 Million records. This dataset is available at grouplens.org as a zip file.[2]

The objective of this project is to create a Netflix style movie recommendation system that produces a Root Mean Square Error (RMSE) result below 0.86490. This project contains two separate ensemble modeling approaches, both of which meet the objective of this project. The first modeling approach used a combination of Principal Component Analysis (PCA) and the base model from the edX class to achieve an RMSE of 0.854. The second modeling approach used a Matrix Factorization ensemble to achieve an RMSE of 0.780.

Overall project method of approach :

1. Verify the data to be used in the analysis
2. Develop a deeper understanding of the data
3. Recreate the class model to be used as a base
4. Look for some simple extensions of the base model
5. Develop a model based on Principal Component Analysis (PCA)
6. Develop a model based on Matrix Factorization
7. Validate the models
8. Evaluate the models and develop a set of conclusions
9. Document the results

Both models worked well. Matrix Factorization using the recosystems package, which is particularly targeted for this solution, was by far the superior model from a results, simplicity, run time, and memory management perspective.

[1]https://www.edx.org/professional-certificate/harvardx-data-science
[2]http://files.grouplens.org/datasets/movielens/ml-10m.zip

2 Setup

The inclusion of a Setup section, and specifically showing the code for this section, is unusual for a report. It is being included because replication and validation of the underlying code are a very important component of this project. The sole purpose of this section is to assist in running the model and duplicating results. The entire section can be skipped if the reader is not interested in running the underlying code.

2.1 General

1. Memory - Running this project will require 32 GB
2. Runtime - Knitting the entire project requires about 2 hours and 45 minutes on a desktop PC with a Intel Core i5 Processor.
3. Memory management - Keeping the project from exceeding available memory was a challenge. In order to reduce the problem, key results are written to the working directory and read back in to memory when needed.
4. Run-size - the run size parameter, discussed below, can be used to run the model quickly and with less memory but will not produce good RSME results.

2.2 Libraries, Functions, and Run Size

This section loads the libraries used in this project, creates the functions used throughout the code. All functions used have been consolidated in this code chunk except for the function that creates the test and training data sets. That function is in a separate code chunk at the beginning of the Modeling Section of this Report.

This section also contains a run_size parameter that the analyst can set when evaluating or testing. Setting a small run_size is useful for quickly testing and running the project. The run_size parameter impacts the model runs, the Data Verification section is not controlled by run_size. A full run requires about 3 hours on my computer.

Run size *MUST* be set to 1 for valid results. Full evaluation of the validation set will not happen unless run size is set to one. All RMSE numbers are significantly worse with small run sizes. Model Validation will not work properly with a run_size set below 1.

```r
#set global options for code chunks to no messages or warnings
knitr::opts_chunk$set(
    message = FALSE,
    warning = FALSE,
    echo = FALSE
)

#Load the libraries
library(tidyverse)
library(caret)
library(knitr)
library(kableExtra)
library("FactoMineR")
library("factoextra")
library(recosystem)

#turn off scientific notation and set number of digits to print
```

```r
options(scipen = 999, digits = 8)

#Function to calculate Root Mean Square Error
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

#create a "not in" function
`%notin%` <- Negate(`%in%`)

#create a cleanup function to remove unneeded variables from memory
cleanup <- function(keep = ""){
  # set up a default keep list
  keep_default <- c("RMSE","rcst","create_test_and_train",
                    "run_size", "%notin%", "cleanup", "rmse_results",
                    "start_time", "update_results_table",
                    "display_results_table")
  # concatenate the user specified keep list to the default
  keep <- c(keep_default, keep)
  # get a list of all objects in the global environment
  all_objects <- ls(envir = .GlobalEnv)
  # set up the index for items to be deleted
  ind <- all_objects %notin%  keep
  # delete the unwanted items
  rm( list = all_objects[ind], envir = .GlobalEnv)
}

# This function simplifies the update of the rmse_results after each model run.
update_results_table  <- function(results, method, type) {
  # the <<- operator updates the global rmse_results object by reference
  rmse_results <<- bind_rows(
    rmse_results,
    tibble(
      Method = method,
      Type = type,
      RMSE = results
    )
  )
}

# This function simplifies the display of the rmse_results
# It also uses conditional formatting to identify the quality of the results
display_results_table <- function(caption) {
  rmse_results %>%
    mutate(RMSE = cell_spec(RMSE,
                            "latex",
                            color = ifelse(RMSE > 0.89999, "blue",
                                           ifelse(RMSE > 0.86490,
                                                  "orange",
                                                  "red")))) %>%
    kable("latex",
          escape = F,
          booktabs = T,
          linesep = "",
          caption = caption) %>%
    kable_styling(latex_options = c("striped", "hold_position")) %>%
    footnote(general = "BLUE:needs work   ORANGE:getting there   RED:exceeds target",
            general_title = "Color Key ")
}

# The rcst function is used in the PCA analysis
# The rcst (reconstruct) function converts the normalized y matrix back to
# a set of predicted ratings. It essentially reverses the process used to
# originally construct the y matrix.
rcst <- function(m){
  m <- sweep(m, 1,  y_row_means2, "+")
  m <- sweep(m, 2, y_col_means, "+")
  m <- sweep(m, 1, y_row_means1, "+")
}

#run_size controls run size throughout the project.
#For a full formal analysis, set the size to 1.0
#For a quick test run that completes in under 10 minutes,
#(without the initial data download - turn that off)
#  set the run_size to 0.001
#RMSE results will be significantly worse for small test runs
run_size <- 1
```

2.3 Data Load

This code for this section was provided as part of the course and is essentially unchanged. It was only modified to reflect the current version of R that I am using (R version 4.0.2) and to save the data sets, so the code will not be displayed in the output pdf document

Running this chunk of code requires considerable time. After the initial run, the edx and validation dataframes are saved to the working directory as rda files. If desired the options for this chunk can then be set to "eval = FALSE" . On my computer, this saves about 50 minutes of run time.

3 Data Verification

In this section we will perform basic checks of the data and the size of the datasets. We will also take an initial look at the characteristics of the data in order to gain insight before beginning the more detailed modeling and analysis steps.

3.1 Check the size and composition of the edx data set

We should see 9,000,055 obs. of 6 variables

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474..
```

Table 1: edx Data Set - First 15 rows

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	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
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy
1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War
1	362	5	838984885	Jungle Book, The (1994)	Adventure|Children|Romance
1	364	5	838983707	Lion King, The (1994)	Adventure|Animation|Children|Drama|Musical
1	370	5	838984596	Naked Gun 33 1/3: The Final Insult (1994)	Action|Comedy
1	377	5	838983834	Speed (1994)	Action|Romance|Thriller
1	420	5	838983834	Beverly Hills Cop III (1994)	Action|Comedy|Crime|Thriller
1	466	5	838984679	Hot Shots! Part Deux (1993)	Action|Comedy|War
1	520	5	838984679	Robin Hood: Men in Tights (1993)	Comedy
1	539	5	838984068	Sleepless in Seattle (1993)	Comedy|Drama|Romance

Table 2: Validation Data Set - First 15 rows

userId	movieId	rating	timestamp	title	genres
1	231	5.0	838983392	Dumb & Dumber (1994)	Comedy
1	480	5.0	838983653	Jurassic Park (1993)	Action|Adventure|Sci-Fi|Thriller
1	586	5.0	838984068	Home Alone (1990)	Children|Comedy
2	151	3.0	868246450	Rob Roy (1995)	Action|Drama|Romance|War
2	858	2.0	868245645	Godfather, The (1972)	Crime|Drama
2	1544	3.0	868245920	Lost World: Jurassic Park, The (Jurassic Park 2) (1997)	Action|Adventure|Horror|Sci-Fi|Thriller
3	590	3.5	1136075494	Dances with Wolves (1990)	Adventure|Drama|Western
3	4995	4.5	1133571200	Beautiful Mind, A (2001)	Drama|Mystery|Romance
4	34	5.0	844416936	Babe (1995)	Children|Comedy|Drama|Fantasy
4	432	3.0	844417070	City Slickers II: The Legend of Curly's Gold (1994)	Adventure|Comedy|Western
4	434	3.0	844416796	Cliffhanger (1993)	Action|Adventure|Thriller
5	85	3.0	857912791	Angels and Insects (1995)	Drama|Romance
5	171	3.0	857912492	Jeffrey (1995)	Comedy|Drama
5	232	3.0	857912933	Eat Drink Man Woman (Yin shi nan nu) (1994)	Comedy|Drama|Romance
5	242	3.0	857912738	Farinelli: il castrato (1994)	Drama|Musical

```
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "St"..
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci"..
##  - attr(*, ".internal.selfref")=<externalptr>
```

3.2 Check the size and composition of the validation data set

We should see 999,999 obs. of 6 variables. After verifying the correct size and structure we will drop the validation set from memory and we will not use it again until we get to final model validation.

```
## Classes 'data.table' and 'data.frame':   999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920..
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone "..
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Come"..
##  - attr(*, ".internal.selfref")=<externalptr>
```

3.3 Understanding the data

The data has only 6 variables. Each row represents an individual user's rating of a specific movie. The data is tidy. The problem can be formulated as trying to predict a rating, the unknown variable, given the other 5 known variables.

The 5 known variables can be categorized along 4 dimensions:

1. Movies - which includes both movie ID and title.
2. Users - listed by ID number
3. Genres - which can be viewed as describing both movies and user preferences
4. Time - the timestamp reflects the time of the user rating

We are not trying to predict the ratings of unknown users or unknown movies. This means that the size of the problem is essentially fixed and could be represented by a matrix that had m columns and u rows. Representing movies and users respectively.

```
## The matrix of 69878 users and 10677 movies contains 746087406 cells.
## Since edx contains 9000055 observations, the matrix is 1.2063004 % occupied.
```

A 1.2% occupancy rate means that we have a very sparse matrix. "You can think of the task of a recommendation system as filling in the NAs"[3] in the matrix. Since we know the users and movies contained in the validation set, we have bounded the problem. We don't know which empty cells will be required to complete the validation, so we need to be able to predict any cell in our very sparse matrix.

3.3.1 Movies

Intuitively, movies are a key driver of the rating. Popularity and ratings of movies varies greatly. From the histogram of ratings (Figure 1) we can see that the most common rating is a 4 and the second most common rating is a 3. Half-star ratings are allowed but they are not given as often as full star ratings.

3.3.1.1 Movie popularity If judged by the number of ratings, varies widely. From the histogram shown in Figure 2, we can see that there are over 100 movies with only 1 rating, while there are also over 100 movies with over 10,000 ratings.

The most popular movies in the database (judged by number of ratings) are shown in Table 3.

3.3.1.2 Individual Movie Ratings For any given movie, there are a wide range of ratings. As an example, we will look at the 5th most popular movie in our list, The Shawshank Redemption (movieId 318, Figure 3).

```
## The Shawshank Redemption received 28015
## The mean rating was 4.4551312
## The standard deviation was 0.71702267
```

3.3.2 Users

Users are the other critical component of our analysis. The movie ratings that we saw above were obviously given by users. Judging by the histogram shown in Figure 4, some users are much more active than others.

[3]Irizzary, Rafael. 2019. Introduction to Data Science. pg 639

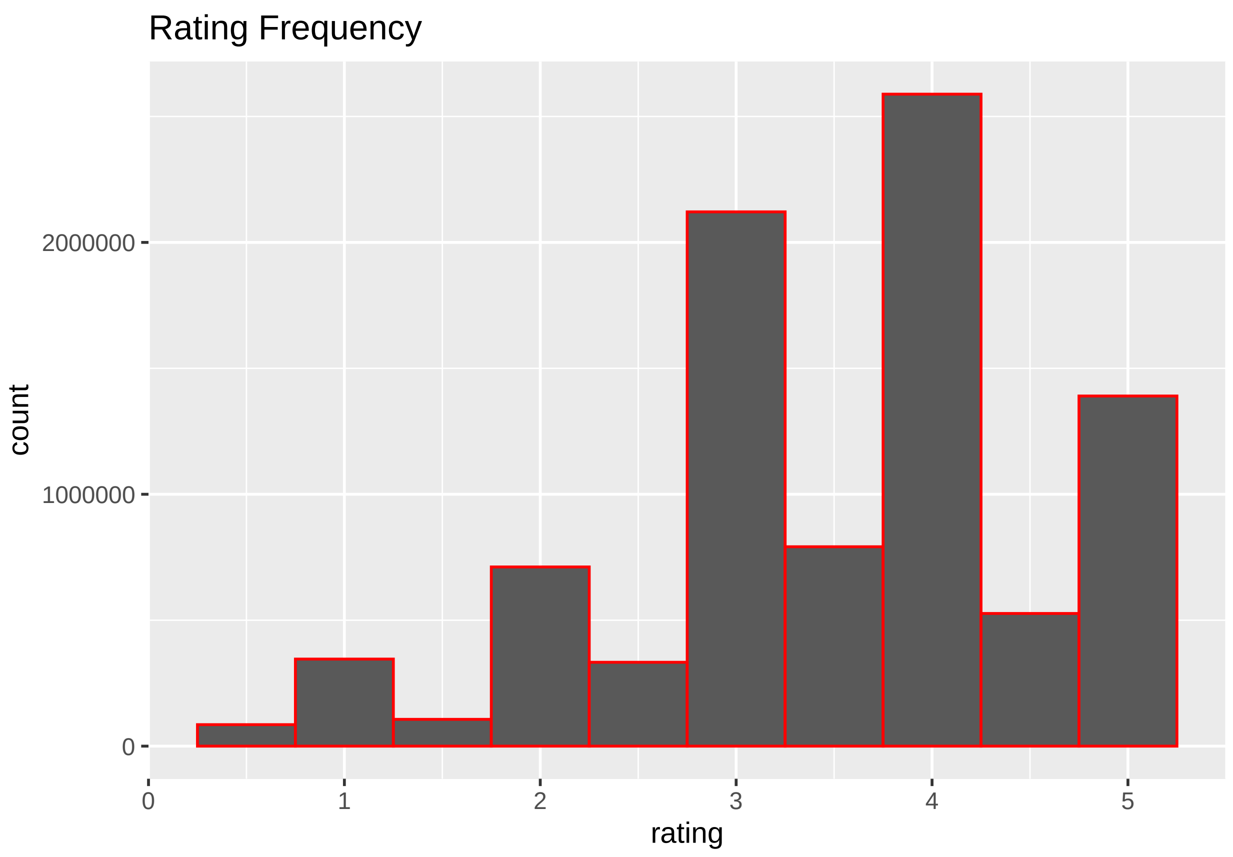

Figure 1: Rating Frequency

Table 3: Most Popular Movies

movieId	title	popularity
296	Pulp Fiction (1994)	31362
356	Forrest Gump (1994)	31079
593	Silence of the Lambs, The (1991)	30382
480	Jurassic Park (1993)	29360
318	Shawshank Redemption, The (1994)	28015
110	Braveheart (1995)	26212
457	Fugitive, The (1993)	25998
589	Terminator 2: Judgment Day (1991)	25984
260	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672
150	Apollo 13 (1995)	24284
592	Batman (1989)	24277
1	Toy Story (1995)	23790
780	Independence Day (a.k.a. ID4) (1996)	23449
590	Dances with Wolves (1990)	23367
527	Schindler's List (1993)	23193

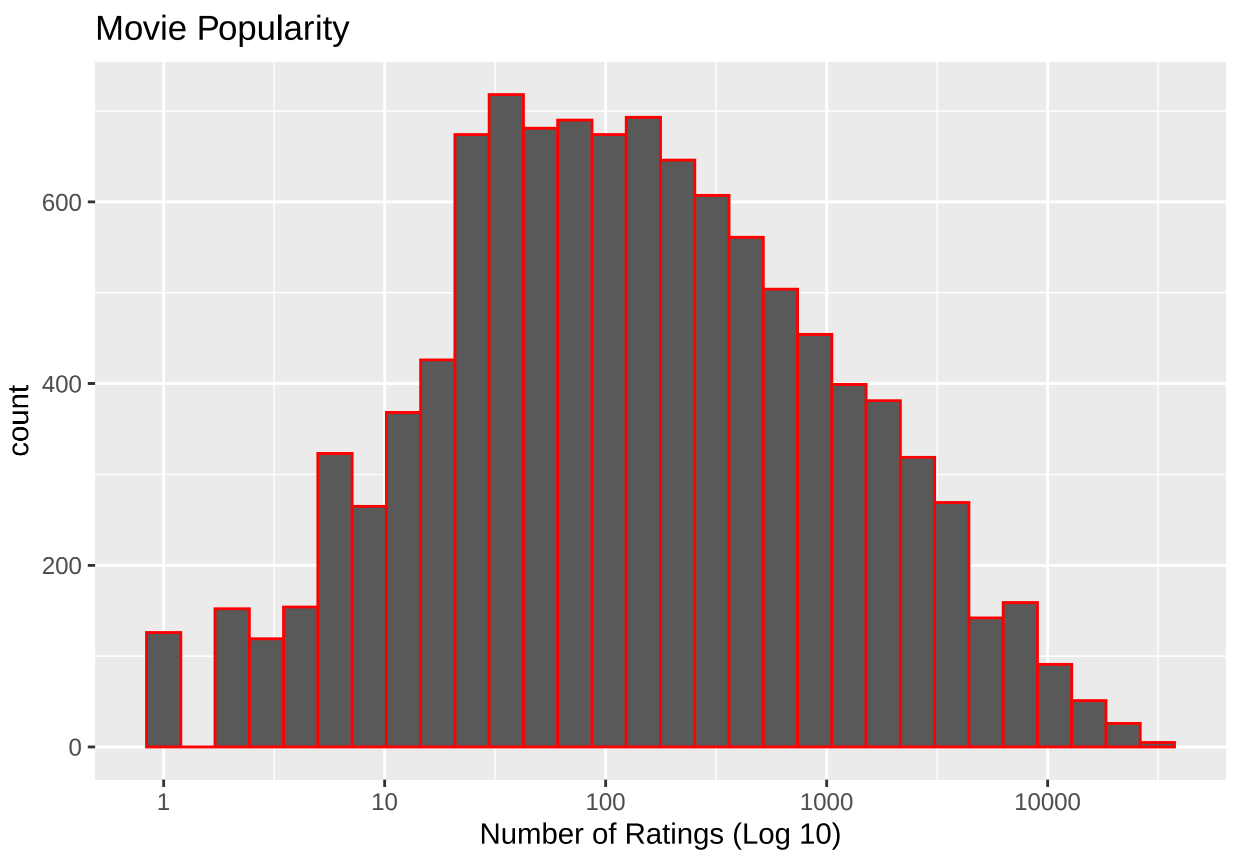

Figure 2: Movie Popularity

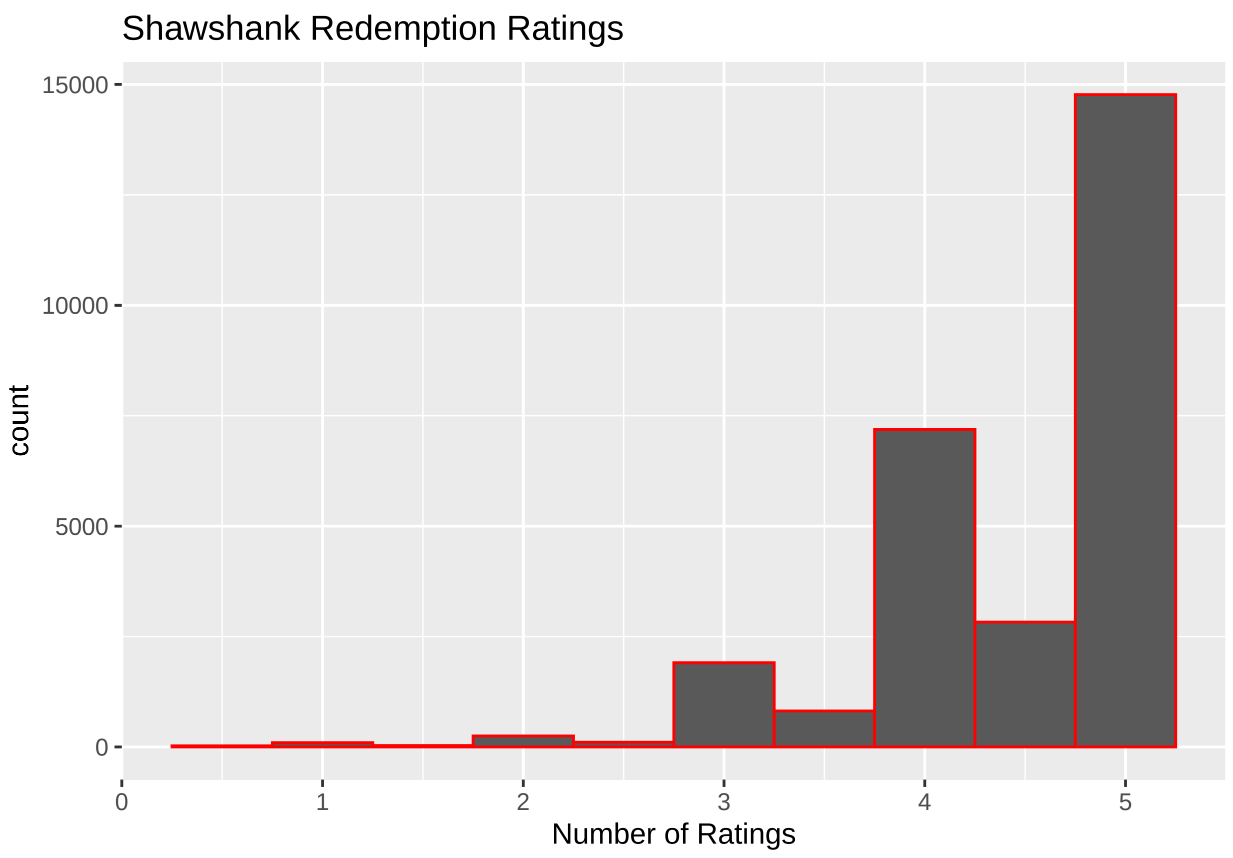

Figure 3: Movie Ratings Example

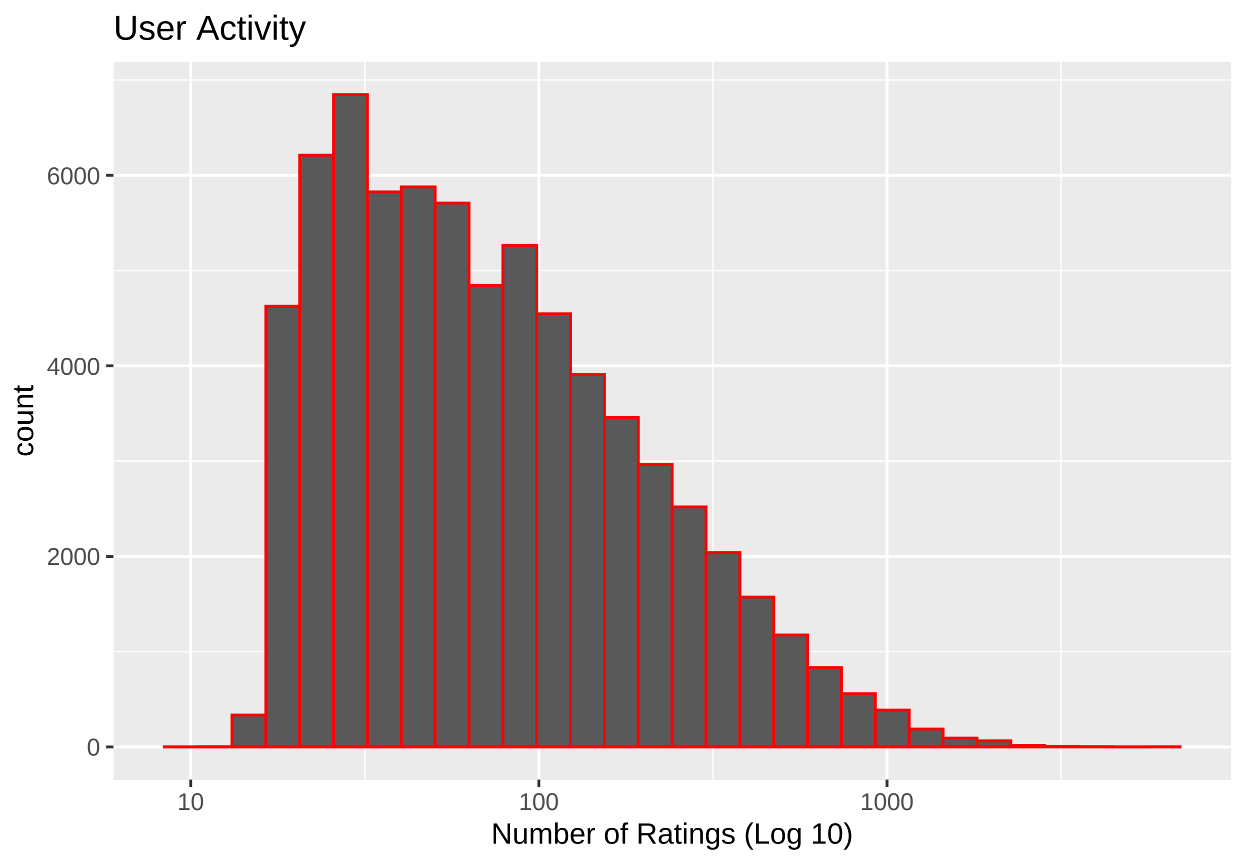

Figure 4: User Rating Activity

3.3.2.1 Individual User Ratings For any given user, there are a wide range of ratings. As an example, we will look at a couple of random user in our list. We can see that their ratings profiles are quite different. The first user, user 318 shown in Figure 5, is very active and tends to rate movies fairly high and gives an unexpected number of half star ratings. User 831 (Figure 6) is less active, also tends to rate most movies fairly highly but never gives half star ratings.

Fairly high movie ratings are the norm with an average rating of 3.65. Not surprisingly, users tend to watch movies they like more frequently than movies they don't like. Users pick movies based on popularity, genres and recommendations.

```
## This user produced 565 ratings.
## The mean rating was 3.6451327
## The standard deviation was 1.123649
```

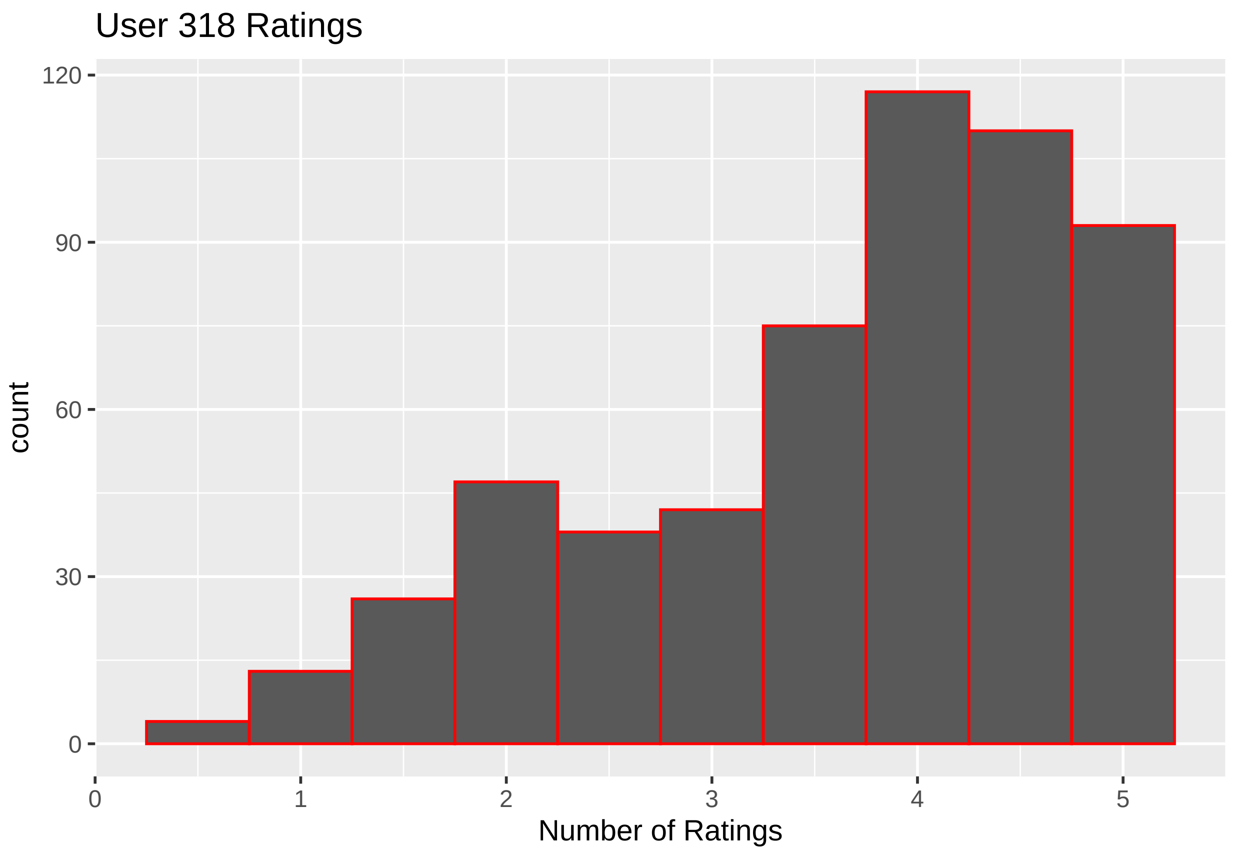

Figure 5: User Ratings Example

```
## This user produced 83 ratings.
## The mean rating was 4.060241
## The standard deviation was 1.2527959
```

3.3.3 Genres

Genres can be looked at from two perspectives, the user perspective and the movie aggregate impact. The user perspective is intuitive. Some users prefer one genre over another. The movie aggregate impact implies

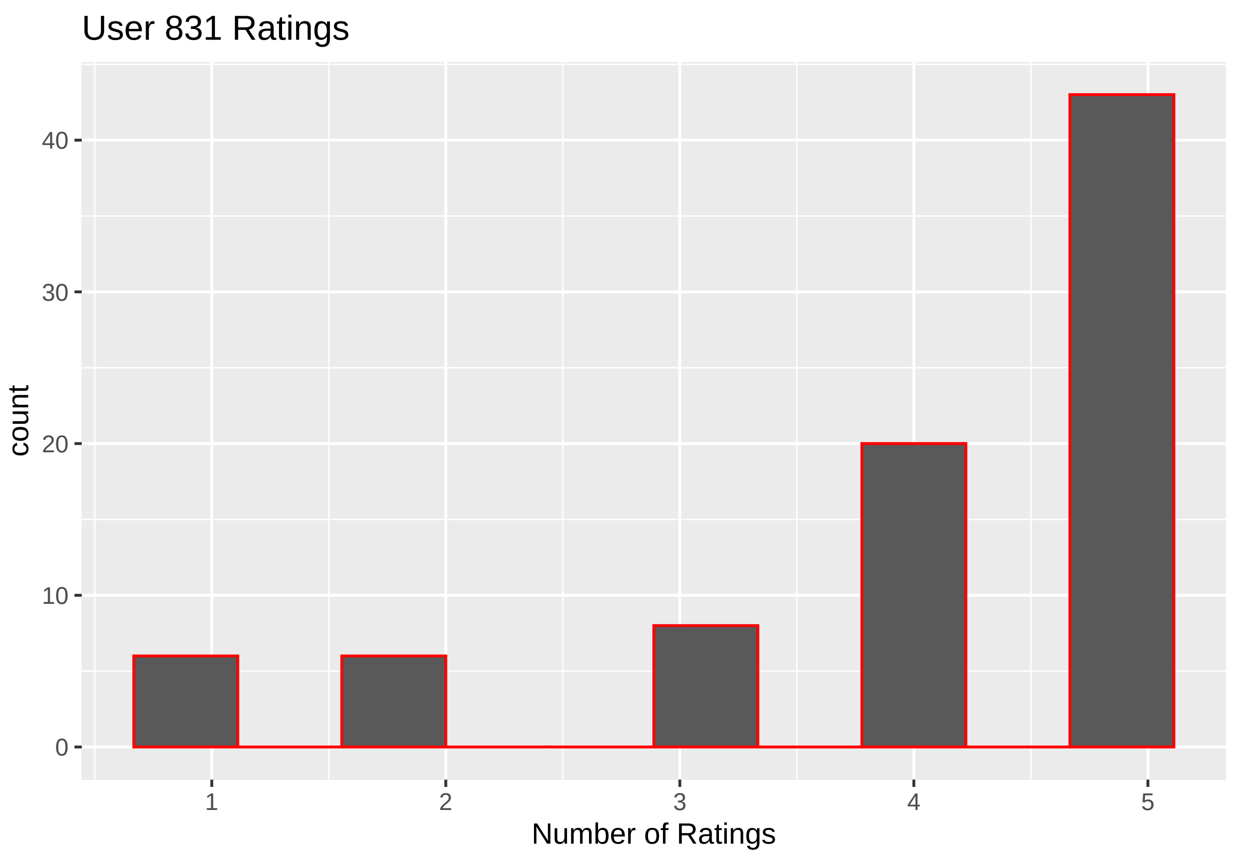

Figure 6: User Ratings Example 2

that some genres are more popular (preferred by more people) than others, and in aggregate impact the movie rating.

Both of these perspectives could be important to our model.

3.3.3.1 The User Perspective Users display distinct preferences in genres. We will use our two example users to demonstrate user genre preference. The first plot (Figure 7) is for a single user and shows the genres ratings ordered by preference. The second plot (Figure 8) adds another user to the plot and demonstrates that our two users have significantly different preferences in genres.

```
## For user 318, there are 218 combinations of genres that were rated.
```

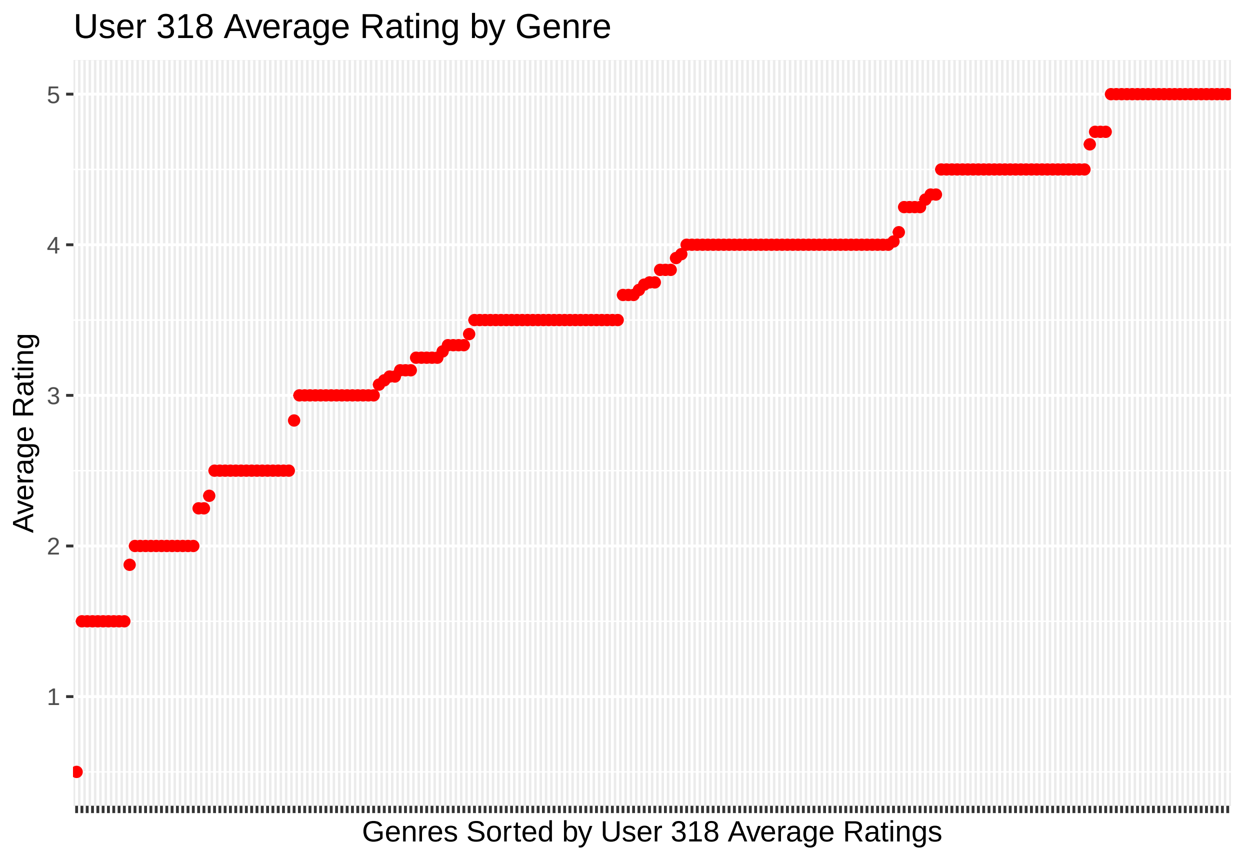

Figure 7: Genre Preference Example

3.3.3.2 The Movie Aggregate Impact The next two tables (Tables 4 and 5) and plots (Figures 9 and 10) give us a better look at the genres data. The two tables show us the genres combinations with the highest and lowest average ratings. Just looking at the means, there appears to be a significant difference in the genres.

Given that there appears to be an impact by genre, the question is whether to look at the genre combinations as seen in the data or to look at the individual simple genres such as “Action”, “Adventure”, “Comedy”, etc. In order to work with the simple genres, for each movie, I split the compound genre into it’s components and then assigned the movie rating average to each genre that was present and a zero to the genres that weren’t present. This is a very simplistic approach.

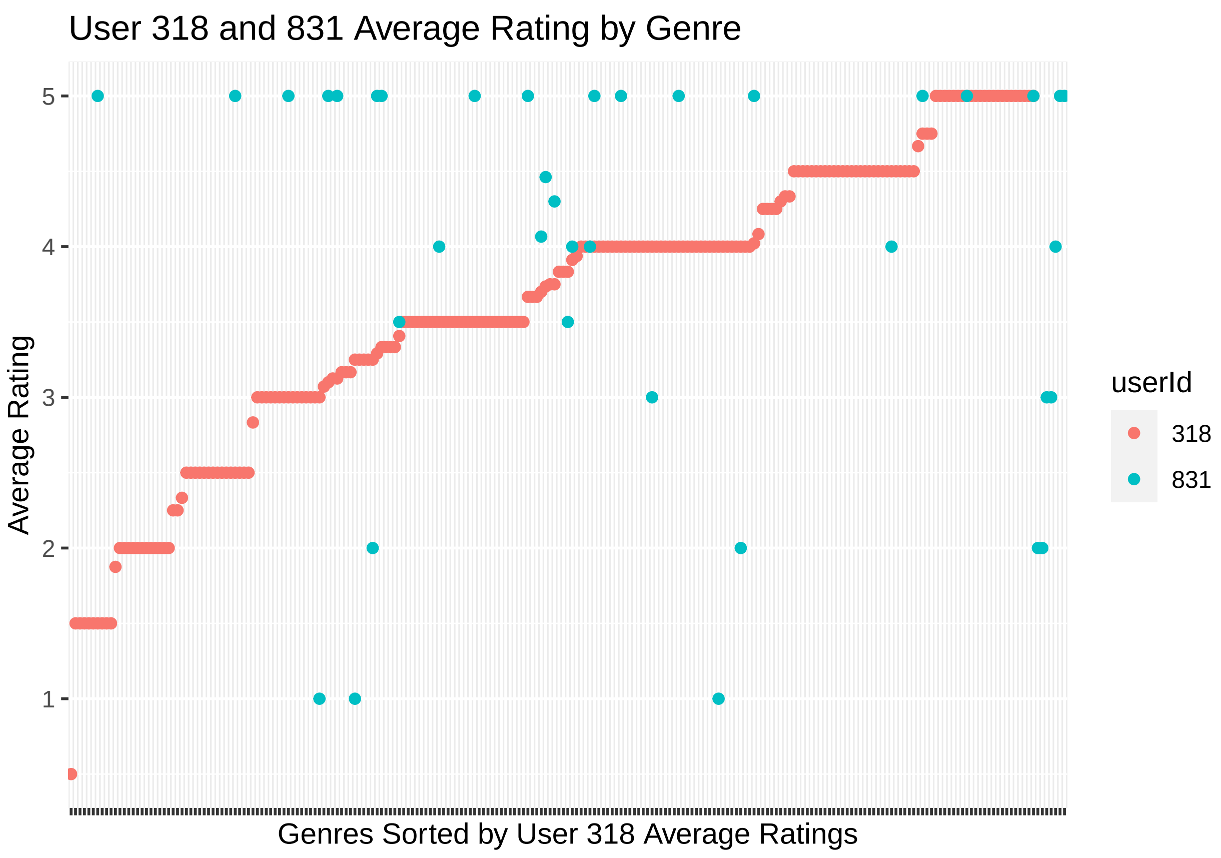

Figure 8: Genre Preference Example 2

Table 4: Highest Rated Genres

genres	avg_rating
Animation|IMAX|Sci-Fi	4.7142857
Drama|Film-Noir|Romance	4.3041151
Action|Crime|Drama|IMAX	4.2970676
Animation|Children|Comedy|Crime	4.2754290
Film-Noir|Mystery	4.2394790
Crime|Film-Noir|Mystery	4.2168032
Film-Noir|Romance|Thriller	4.2164696
Crime|Film-Noir|Thriller	4.2101569
Crime|Mystery|Thriller	4.1989811
Action|Adventure|Comedy|Fantasy|Romance	4.1955568
Crime|Thriller|War	4.1712731
Film-Noir|Mystery|Thriller	4.1642982
Adventure|Drama|Film-Noir|Sci-Fi|Thriller	4.1483843
Adventure|Animation|Children|Comedy|Sci-Fi	4.1479173
Adventure|Comedy|Romance|War	4.1279916
Adventure|Animation|Children|Comedy|Romance|Sci-Fi	4.1228070
Comedy|Drama|Romance|Sci-Fi	4.1179376
Action|Crime|Drama|Film-Noir|Mystery	4.1155938
Animation|Drama|War	4.1126083
Action|Drama|Thriller|War	4.1093750

Also, I wanted to understand the confidence intervals for both the simple genres and the genre combinations. The two error bar plots that follow show this information. From a prediction perspective, what we would really like to see is separation between the means and some non-overlapping confidence intervals.

While there is differentiation in the means, the confidence interval plots by genres show very wide confidence intervals that completely overlap almost all genre means. I was hoping that simple genres might offer more differential in rating, but they cluster tightly around the overall movie average of 3.6. This is probably caused by using an oversimplified method for assigning values to the simple genres. Making the simple genres approach work may require using effect coding (-1 and 1) instead of "dummy coding", and to take the additional step of centering the rating variable.

Compound genres show a bit more variation and are much easier to work with, so we will use the compound genres for our modeling. With compound genres, there is still the issue of significant overlap of the confidence intervals, but we have more dispersion in the means.

3.3.4 Timestamp

Although both the class examples and the Bellkor examples discuss a time impact, do to the desire to limit scope, we will not consider time impact in this analysis.

4 Modeling

The modeling approach was to:

1. Create a function to consistently and reliably produce clean training and test sets

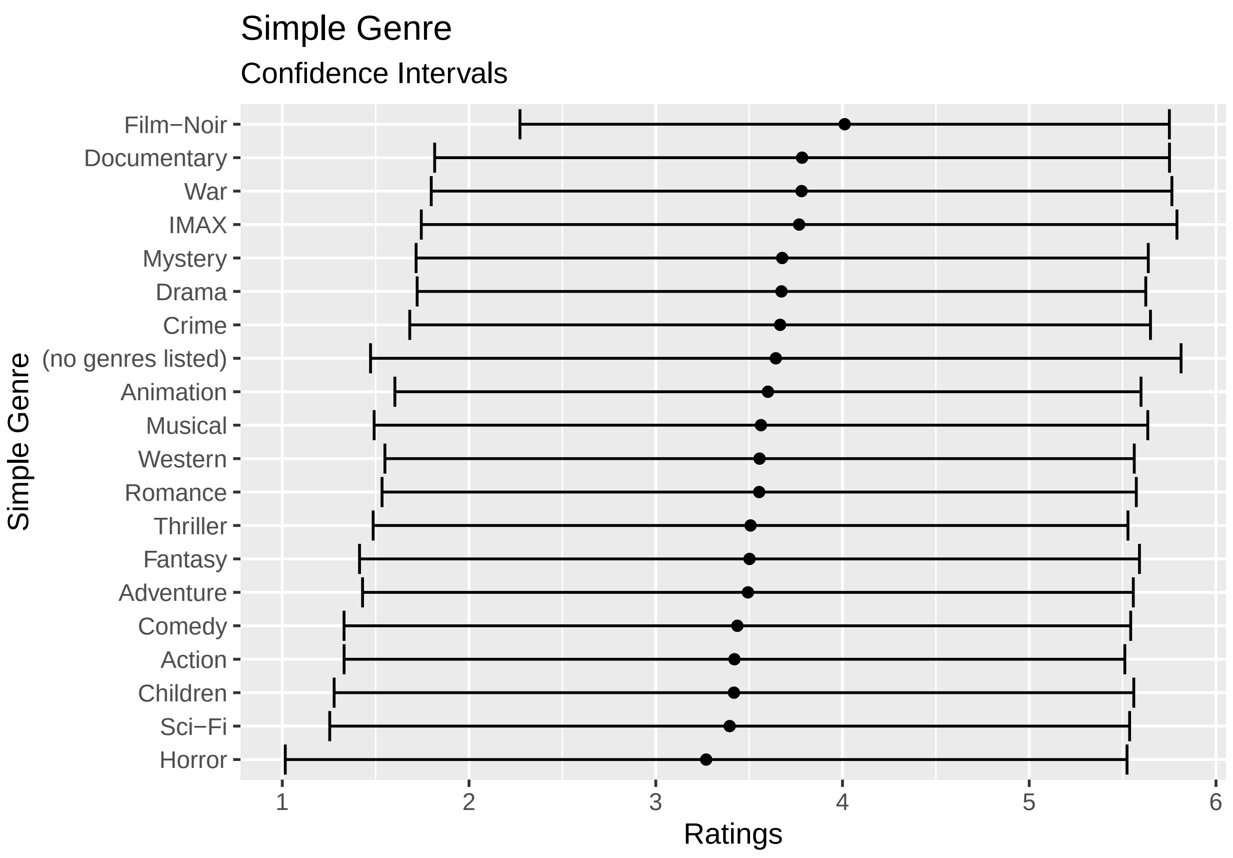

Figure 9: Genres Confidence Interval 1

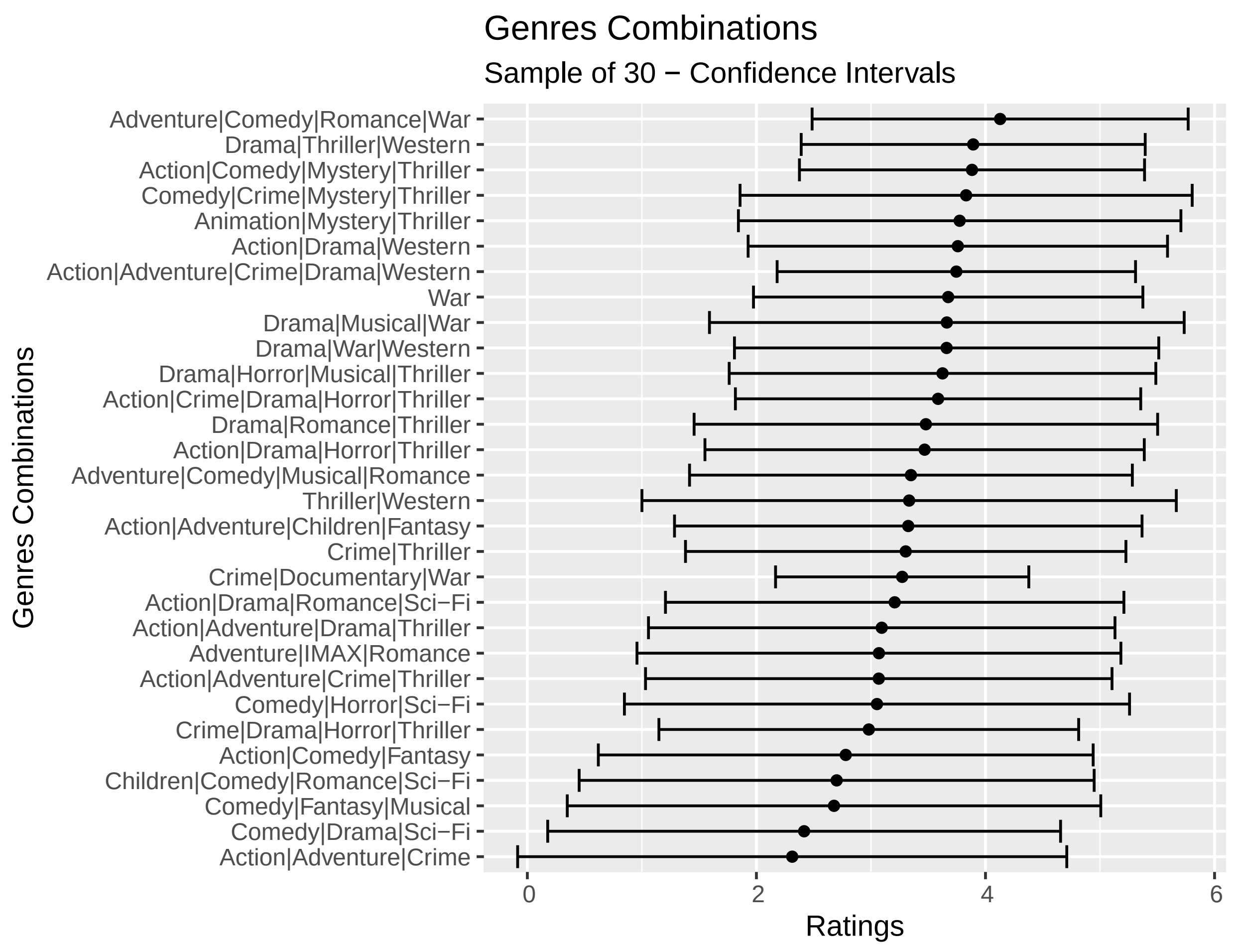

Figure 10: Genres Confidence Interval 2

Table 5: Lowest Rated Genres

genres	avg_rating
Adventure|Comedy|Fantasy|Sci-Fi	2.2400000
Action|Adventure|Fantasy|Thriller	2.2063348
Action|Comedy|Drama|Romance	2.1904762
Crime|Sci-Fi|Thriller	2.1814122
Adventure|Comedy|Drama|Fantasy	2.1607143
Action|Adventure|Children|Comedy|Mystery	2.1540984
Action|Adventure|Children|Comedy|Fantasy|Sci-Fi	2.0561441
Action|Children	2.0441101
Children|Fantasy|Sci-Fi	1.9576271
Action|Horror|Mystery|Sci-Fi	1.9285714
Adventure|Animation|Children|Fantasy|Sci-Fi	1.9247467
Action|Adventure|Children	1.9150485
Action|Children|Comedy	1.9102317
Action|Adventure|Drama|Fantasy|Sci-Fi	1.9035088
Adventure|Drama|Horror|Sci-Fi|Thriller	1.7511521
Action|Drama|Horror|Sci-Fi	1.7500000
Comedy|Film-Noir|Thriller	1.6428571
Action|Horror|Mystery|Thriller	1.6070336
Action|Animation|Comedy|Horror	1.5000000
Documentary|Horror	1.4491115

2. Recreate the class model to be used as a base
3. Look for some simple extensions of the base model
4. Develop a model based on Principal Component Analysis (PCA)
5. Develop a model based on Matrix Factorization

The modeling process was iterative and resulted in additional data analysis as model results were evaluated and new insights were drawn. That additional data analysis is contained in this section and is introduced as part of the modeling process.

In order to provide deeper insight into the actual model, for most of the modeling sections the code chunks will be included as part of this report.

4.1 Train and test set creation

This step creates a function that will be used to build the training set and the test set from the edx object. For this analysis we are using a split data set approach with 80% of the data going to the training set and 20% reserved as the test set. The Validation data loaded in the previous chunk will not be used for any training or testing purposes and is reserved for final validation of the model.

If a run size (run_size) of less than one is set, the function randomly selects the specified proportion of users from the overall edx data set. Once the users are selected, movies with no ratings are also stripped. This creates a much smaller data set to work with for testing and exploratory purposes.

```r
# Function to create test and training data frames from edx
# The function returns a list containing the two data frames.
create_test_and_train <- function(proportion = 1,
                                  save = TRUE,
                                  seed = 831,
                                  data_frame = "default"){
  # The edx data set is the training dataset for this lab.
  # By default(proportion = 1) we will use the full edx data set
  # If proportion is set to a number between zero and one,
  # we will use the proportion specified.
  # if proportion is not between 0 and 1 we will generate an error.
  if (proportion < 0 | proportion > 1){
    stop("Error in create_test_and_train function, proportion value not between 0 and 1.")
  }
  #set the seed value for repeatablity between runs
  set.seed(seed, sample.kind="Rounding")
  # if using R 3.5 or earlier, use `set.seed(seed)` instead

  #allows the dataframe to default to edx or be overridden in the function call
  if(data_frame == "default"){
     edx <- read_rds("edx.rda")
  } else{
    edx <- data_frame
  }

  #keep specified proportion of randomly selected users
  temp <- edx %>% group_by(userId) %>%
    summarize(count = n()) #summarize preserves the grouping
  #randomly select percentage of the userIds to keep
  ind_users <- sample(c(0,1),
                      nrow(temp),
                      replace=TRUE,
                      prob = c(1-proportion, proportion)) %>%
    as.logical()

  selected_users <- temp$userId[ind_users]
  edx_small <- edx %>%
    filter(userId %in% selected_users)

  #strip out movies with 0 reviews
  selected_movies <- edx_small %>%
    group_by(movieId) %>%
    summarize(count = n()) %>%
    filter(count > 0)

  edx_small <- edx_small %>% filter(movieId %in% selected_movies$movieId)

  #build a small test and training set from edx_small
  test_index <- createDataPartition(y = edx_small$rating, times = 1,
                                    p = 0.2, list = FALSE)
  train_set <- edx_small[-test_index,]
  temp <- edx_small[test_index,]
  test_set <- temp %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

  # Add rows removed from validation set back into edx set
  removed <- anti_join(temp, test_set)
  train_set <- rbind(train_set, removed)

  # Save the test and train sets in the working directory if save = TRUE
  if(save == TRUE){
    write_rds(train_set, "train_set.rda")
    write_rds(test_set, "test_set.rda")
  }

  # return test and train data sets as a list
  test_and_train_list <- list(train_set = train_set, test_set = test_set)
}
```

4.2 Base Model from edX Class

The foundation analysis builds a base prediction model that starts by simply predicting based on the average rating of all movies. The model is expanded by adding additional factors to account for the residuals in the forecast prediction. And finally, regularization is introduced to reduce the impact of infrequently rated movies and users with low activity. This model essentially follows the winning BellKor model that was used for the Netflix challenge.

> The Netflix challenge used the typical error loss: they decided on a winner based on the residual mean squared error (RMSE) on a test set. We define $y_{u,i}$ as the rating for movie i by user u and denote our prediction with $\hat{y}_{u,i}$. The RMSE is then defined as:
>
> $$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{u,i}\left(\hat{y}_{u,i} - y_{u,i}\right)^2}$$
>
> with N being the number of user/movie combinations and the sum occurring over all these combinations.
>
> Remember that we can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star, which is not good.
>
> We know from experience that some movies are just generally rated higher than others. This intuition, that different movies are rated differently, is confirmed by data. We can augment our previous model by adding the term b_i to represent average ranking for movie i:
>
> $$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$
>
> Statistics textbooks refer to the bs as effects. However, in the Netflix challenge papers, they refer to them as "bias", thus the b notation.
>
> The general idea behind regularization is to constrain the total variability of the effect sizes. Why does this help? Consider a case in which we have movie $i = 1$ with 100 user ratings and 4 movies $i = 2, 3, 4, 5$ with just one user rating. We intend to fit the model
>
> $$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

Suppose we know the average rating is, say, $\mu = 3$. If we use least squares, the estimate for the first movie effect b_1 is the average of the 100 user ratings, $1/100 \sum_{i=1}^{100} (Y_{i,1} - \mu)$, which we expect to be a quite precise. However, the estimate for movies 2, 3, 4, and 5 will simply be the observed deviation from the average rating $\hat{b}_i = Y_{u,i} - \hat{\mu}$ which is an estimate based on just one number so it won't be precise at all. Note these estimates make the error $Y_{u,i} - \mu + \hat{b}_i$ equal to 0 for $i = 2, 3, 4, 5$, but this is a case of over-training. In fact, ignoring the one user and guessing that movies 2,3,4, and 5 are just average movies ($b_i = 0$) might provide a better prediction. The general idea of penalized regression is to control the total variability of the movie effects: $\sum_{i=1}^{5} b_i^2$. Specifically, instead of minimizing the least squares equation, we minimize an equation that adds a penalty:

$$\frac{1}{N} \sum_{u,i} (y_{u,i} - \mu - b_i)^2 + \lambda \sum_i b_i^2$$

The first term is just least squares and the second is a penalty that gets larger when many b_i are large. Using calculus we can actually show that the values of b_i that minimize this equation are:

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

where n_i is the number of ratings made for movie i. This approach will have our desired effect: when our sample size n_i is very large, a case which will give us a stable estimate, then the penalty λ is effectively ignored since $n_i + \lambda \approx n_i$. However, when the n_i is small, then the estimate $\hat{b}_i(\lambda)$ is shrunken towards 0. The larger λ, the more we shrink.

[@Irizzary, Rafael, 2019. Introduction to Data Science. <https://leanpub.com/datasciencebook>]

The base model that we will be using to start our analysis can be expressed as:

$$\frac{1}{N} \sum_{u,i} (y_{u,i} - \mu - b_i - b_u)^2 + \lambda \left(\sum_i b_i^2 + \sum_u b_u^2 \right)$$

This section of code is essentially the same as provided in the edX class. The key change to this code is contained in the first 3 lines of code and leverages a function to create the test_set and the train_set. The rmse_results object was also slightly modified to indicate whether the results applied to testing or validation, and the update and display of rmse_result was implemented with a function.

For the sake of brevity, I will not show the code or the details of the original analysis, but will run the model to capture the results for this much larger data set. If you are interested in seeing the detail of the initial model development, please reference the Introduction to Data Science textbook.

The results of the base model are summarized in Table 6. The summary shows the improvement achieved by adding or refining different modeling components step by step. We will continue to use this summarization template for the remainder of our analysis.

4.3 Separate Lambdas for Users and Movies

For the models that we have used to date, a single lambda value has been applied to regularize both users and movies. In this section we explore separating the two lambdas to see if there is any improvement in results.

As the results show in Table 7, there is a very slight improvement when separate lambdas are used. The lambda values ended up being up very close (Figure 11 & 12) and the sensitivity of the model to small changes in lambda is not significant. Since our objective is to develop the lowest RSME possible, we stayed with the

Table 6: Base Model Results

Method	Type	RMSE
Just the average	Test	1.0599763488047
Movie Effect Model	Test	0.943684563857829
Movie + User Effects Model	Test	0.866242706667592
Regularized Movie Effect Model	Test	0.943611139110153
Regularized Movie + User Effect Model	Test	0.865497888766143

Color Key
BLUE:needs work ORANGE:getting there RED:exceeds target

Separate Lambda Model even though there is a computational cost and only a very small improvement in RMSE results.

```
#Separate movie and user lambda's
#this is the same as the base model, except for l_m
#l_m provides a separate lambda for movies, user lambda remained simply l
l_m <- lambda
lambdas <- seq(4, 6, 0.1)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l_m))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)
```

```
l <- lambdas[which.min(rmses)]
cat("The best value of user lambda tested was",  l, "\n")
```

```
## The best value of user lambda tested was 5.1
```

```
## The best value of movie lambda tested was 4.7
```

4.4 A look at prediction error

In order to get a better look at error, Figure 13 shows the residual values (actual - prediction) by actual rating. The plot clearly shows the smallest residuals near the movie mean average of 3.6. As we move to

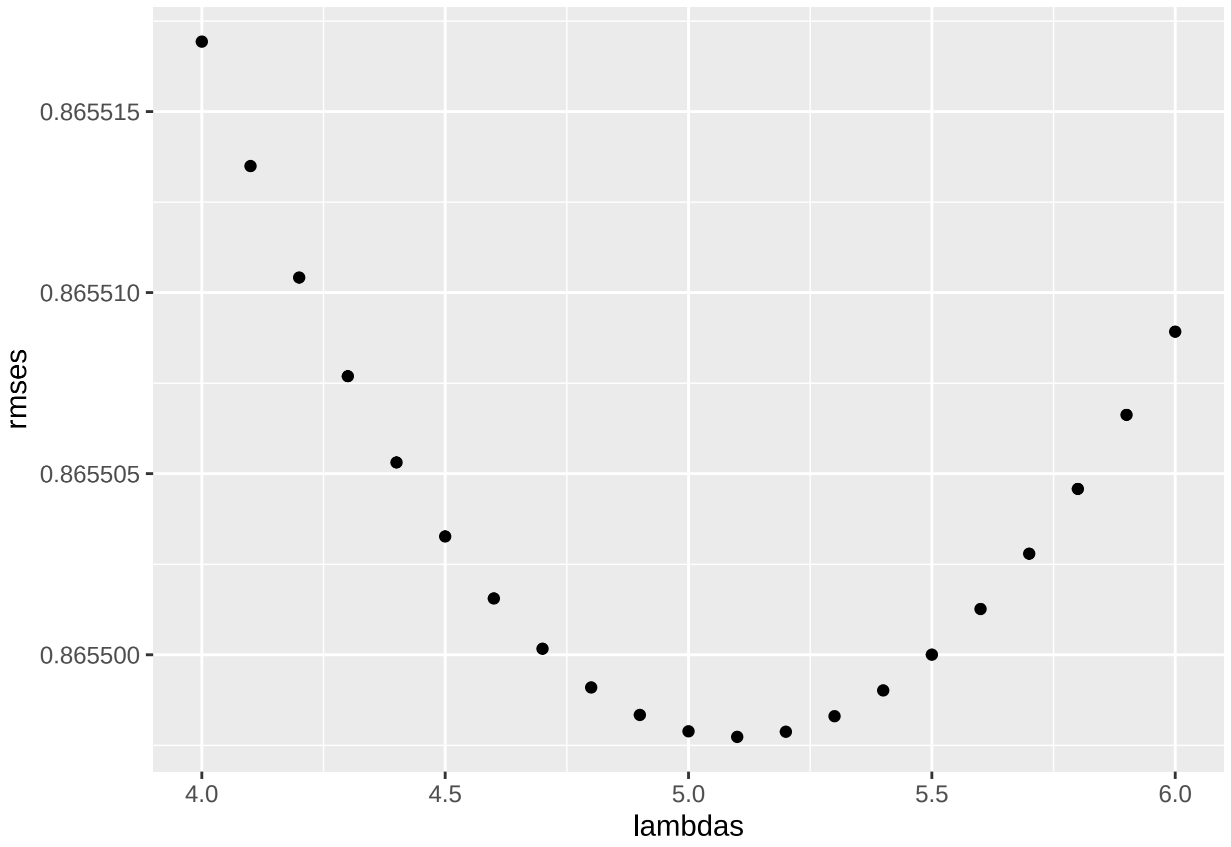

Figure 11: User Lambdas

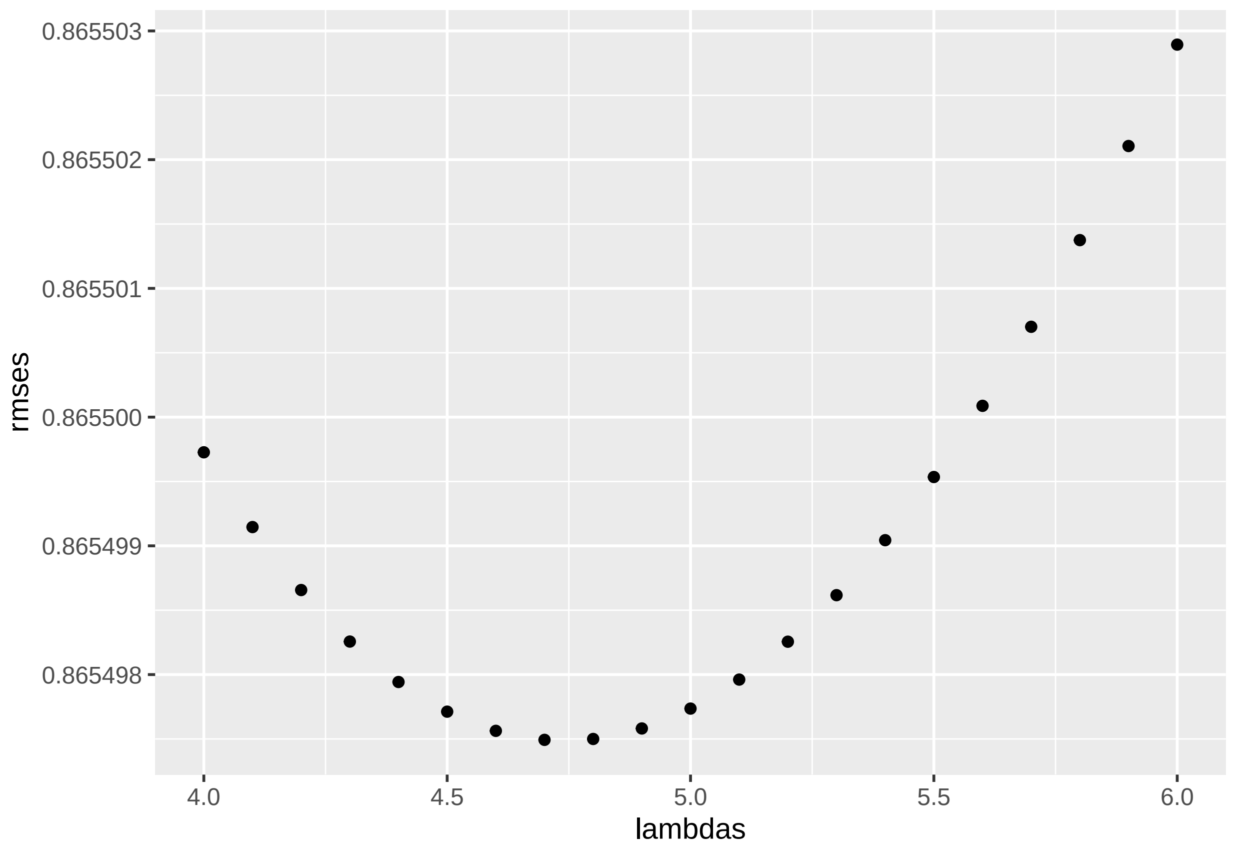

Figure 12: Movie Lambdas

Table 7: RMSE Results

Method	Type	RMSE
Just the average	Test	1.0599763488047
Movie Effect Model	Test	0.943684563857829
Movie + User Effects Model	Test	0.866242706667592
Regularized Movie Effect Model	Test	0.943611139110153
Regularized Movie + User Effect Model	Test	0.865497888766143
Regularized Movie + User Effect Model: Separate Lambdas	Test	0.865497492912466

Color Key
BLUE:needs work ORANGE:getting there RED:exceeds target

either extreme, the residuals seem to increase almost linearly.

A modeling approach that could reduce some of the error at the ends of the range could have a significant impact on the RMSE. Later in this report we will address this opportunity with a post-processing technique that I call "clipping".

4.5 PCA Analysis

As presented in our class textbook[4], matrix Factorization and PCA analysis seemed to offer potential as a model for this project. Up to this point, the models presented required a high level of supervision and analyst interpretation of the data. The prize winning solution developed by the Bellkor Group, relied on a continuation of this approach with a very large and complex ensemble prediction model.[5]

The PCA approach presents the opportunity to let the machine interpret the data and determine the principal components. Because of this advantage, I decided to explore PCA analysis using the FactoMineR package.

The disadvantages of PCA analysis include memory and run-time requirements. These disadvantages will be offset through dimension reduction techniques in this project. Even with dimension reduction, memory and run time requirements still remain long when compared to the base model.

4.5.1 Small initial analysis

The first results were obtained on a very small data set consisting of 0.5% of the users. This was done in order to get a preview of the opportunity on a small set of data that could be run quickly and would allow for easier graphic interpretation. RMSE is not being calculated for this initial analysis.

```r
#load a small data set
df_list <- create_test_and_train(proportion = 0.005, save=FALSE)
train_set <- df_list$train_set

# set y_nmov = the number of movies in the training set (for later use)
y_nmov = train_set %>% group_by(movieId) %>% summarize(obs = n()) %>% nrow()
# set y_nmov = the number of movies in the training set (for later use)
y_nusers = train_set %>% group_by(userId) %>% summarize(obs = n()) %>% nrow()

#convert the dataset into a matrix with movies in columns and users in rows
#y_u,i is the entry in row u and column i => for user u and movie i
y <- train_set %>%
```

[4]Irizzary, Rafael, 2019. Introduction to Data Science.
[5]Bell, Robert, Yehuda Koren and Chris Volinsky. 2008. The BellKor solution to the Netflix Prize

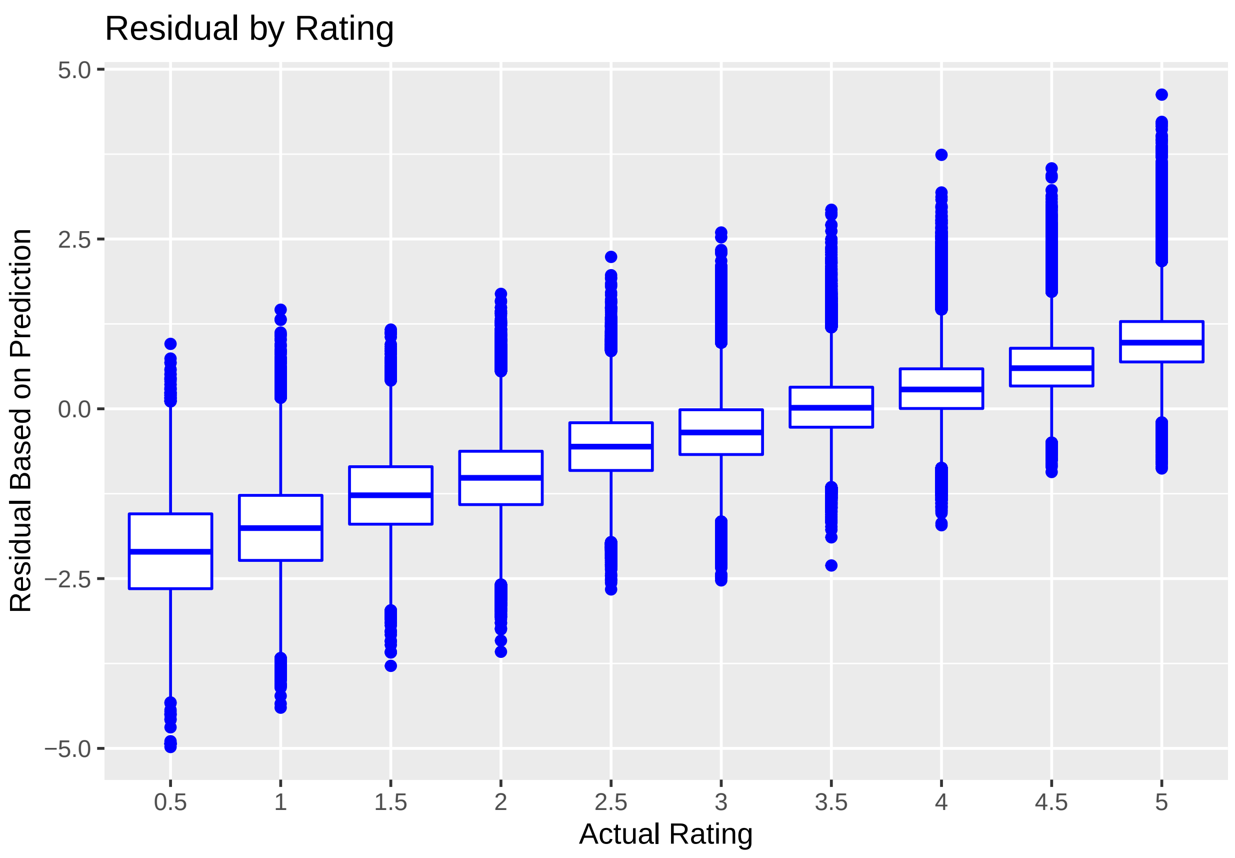

Figure 13: Prediction Error

Table 8: Top 10 Dimensions and Eigenvalues

	eigenvalue	variance.percent	cumulative.variance.percent
Dim.1	2.0108671	2.9777924	2.9777924
Dim.2	1.7044973	2.5241048	5.5018973
Dim.3	1.4553551	2.1551625	7.6570597
Dim.4	1.4152973	2.0958430	9.7529027
Dim.5	1.3644591	2.0205592	11.7734619
Dim.6	1.2784553	1.8932005	13.6666624
Dim.7	1.2055728	1.7852725	15.4519349
Dim.8	1.1862926	1.7567213	17.2086562
Dim.9	1.1041712	1.6351118	18.8437681
Dim.10	1.0343153	1.5316658	20.3754338

```
  select(userId, movieId, rating) %>%
  pivot_wider(names_from = movieId, values_from = rating) %>%
  as.matrix()

movie_titles <- train_set %>%
  select(movieId, title) %>%
  distinct()
# save movieId by column for use during RMSE prediction
movieId_by_col <- colnames(y)

#add rownames and column names to facilitate exploration
rownames(y)<- y[,1]
y <- y[,-1]      #drop the userId column (the rowname now has the userId)
colnames(y) <- with(movie_titles, title[match(colnames(y), movieId)])

#convert to residuals by removing the row and column averages
y_row_means1 <- rowMeans(y, na.rm=TRUE) #save the rowMeans for reconstructing the matrix
y <- sweep(y, 1, rowMeans(y, na.rm=TRUE))
y_col_means <- colMeans(y, na.rm=TRUE)
y <- sweep(y, 2, colMeans(y, na.rm=TRUE))

#Remove NAs and zeroes by sweeping out the row means one more time
y[is.na(y)] <- 0  #make the residuals with NAs = 0
y_row_means2 <- rowMeans(y, na.rm=TRUE)
y <- sweep(y, 1, rowMeans(y))

#Set a value for max principal components
my_ncp <- 10
res.pca <- PCA(y, scale.unit = FALSE, graph = FALSE, ncp = my_ncp)

#We'll use the factoextra R package to help in the interpretation of PCA.

#eigenvalues measure the amount of variation retained by each principal component.
eig.val <- get_eigenvalue(res.pca)
kable(head(eig.val, n=10), caption = "Top 10 Dimensions and Eigenvalues")
```

Table 10 shows the top 10 most significant dimensions and the percent of variance explained. Unfortunately the 10 dimensions only explain a small portion of the variance. This means that more dimensions would need to be included to get good predictions. More dimensions also mean more memory and run time requirements, so we will be working in a sub-optimal environment for this model.

Another way to view importance by dimension is with a Scree plot as shown below. This is just a graphical depiction of the table of eigenvalues.

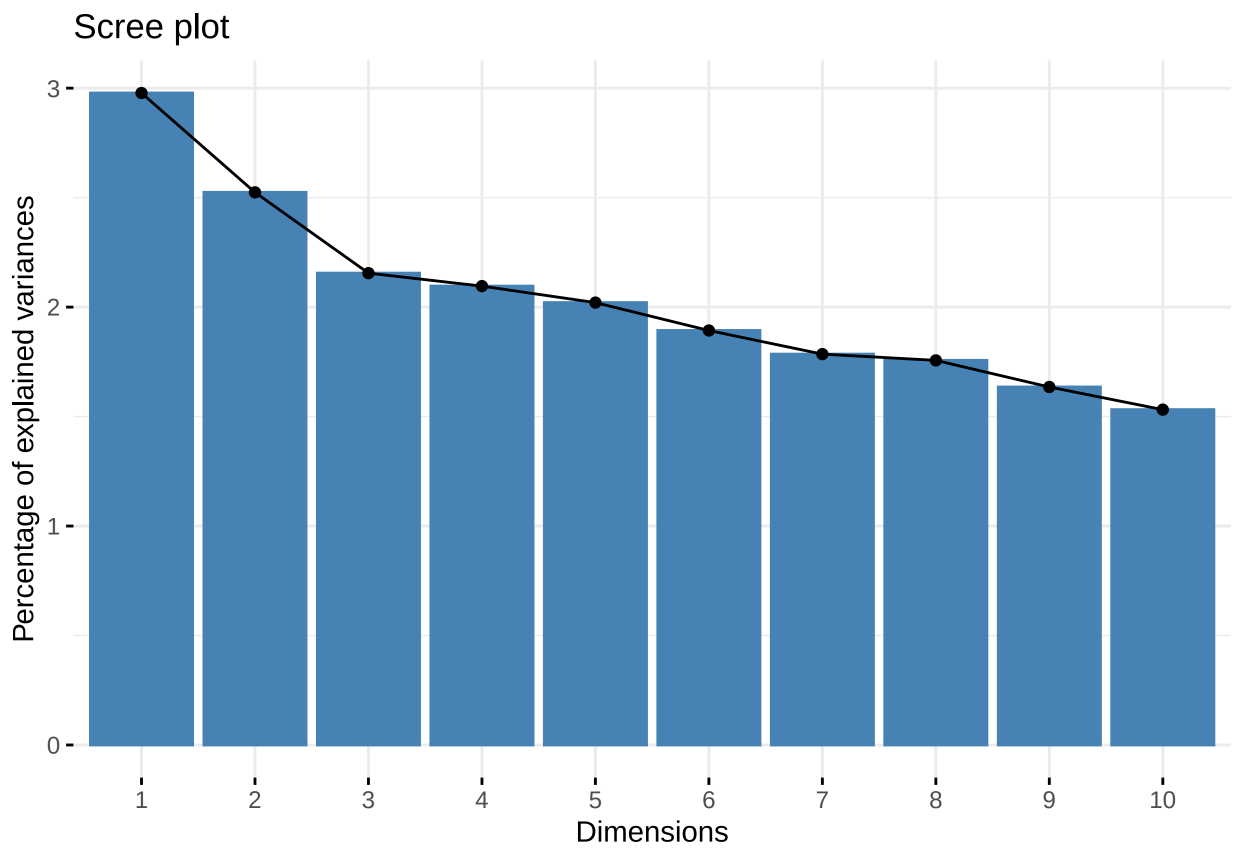

Figure 14: Scree Plot

One of the advantages of the FactoMineR tool for PCA analysis is that it contains some good tools for visualizing the impact of the individual variables (factors), on the principal components. The following two graphs provide an example showing plots of movies and users on the first 2 dimensions.

When looking at these movies, remember that we are working with a very small sample of randomly selected movies. For the full analysis, these plots would look significantly different.

4.5.2 Dimension Reduction

A very simple way to achieve dimension reduction is to only work with the most popular movies. We are going to load and examine the top 15% of the movies based on number of ratings.

```
#set the run size to no more than 15% of the most popular movies
proportion <- if_else(run_size<0.15, run_size, 0.15)

edx <- read_rds("edx.rda")
full_size <- nrow(edx)
```

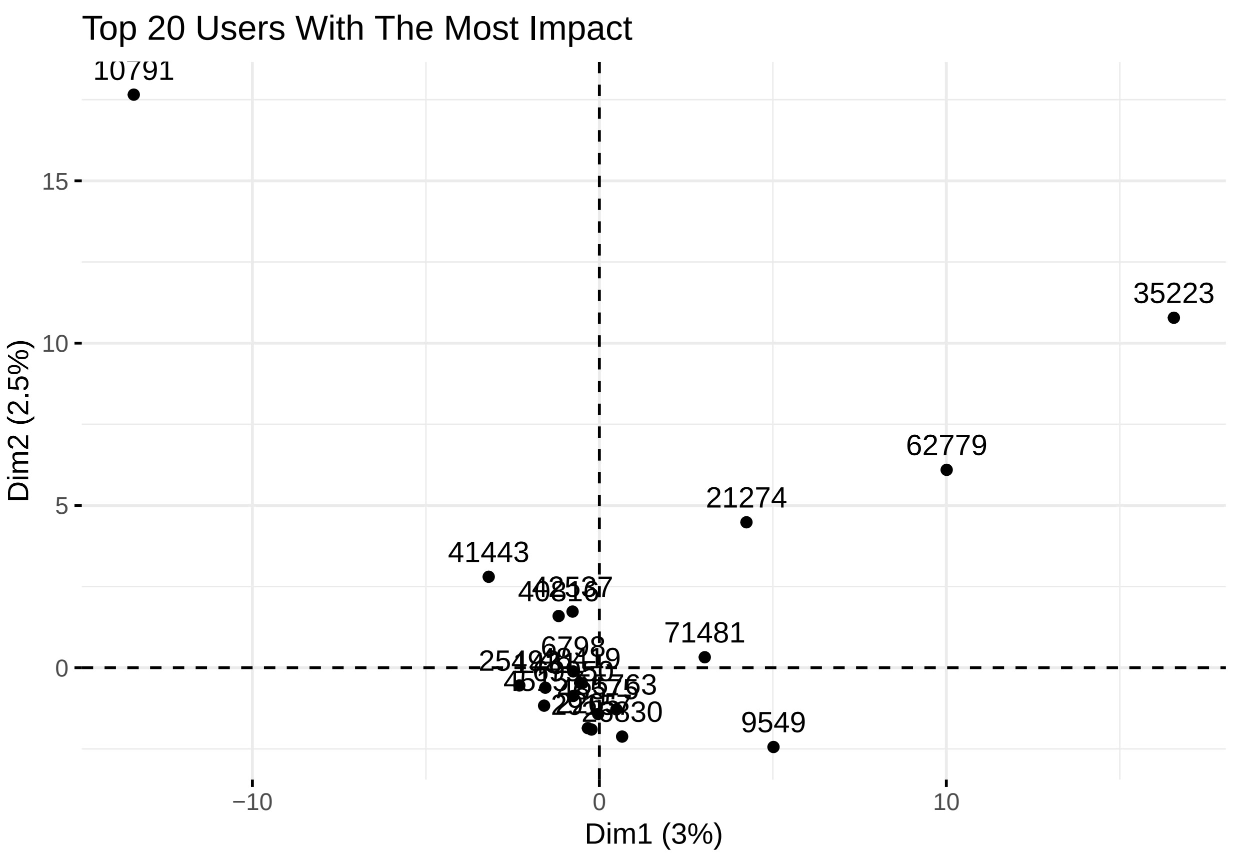

Figure 15: Dimension 1 & 2 User Impact

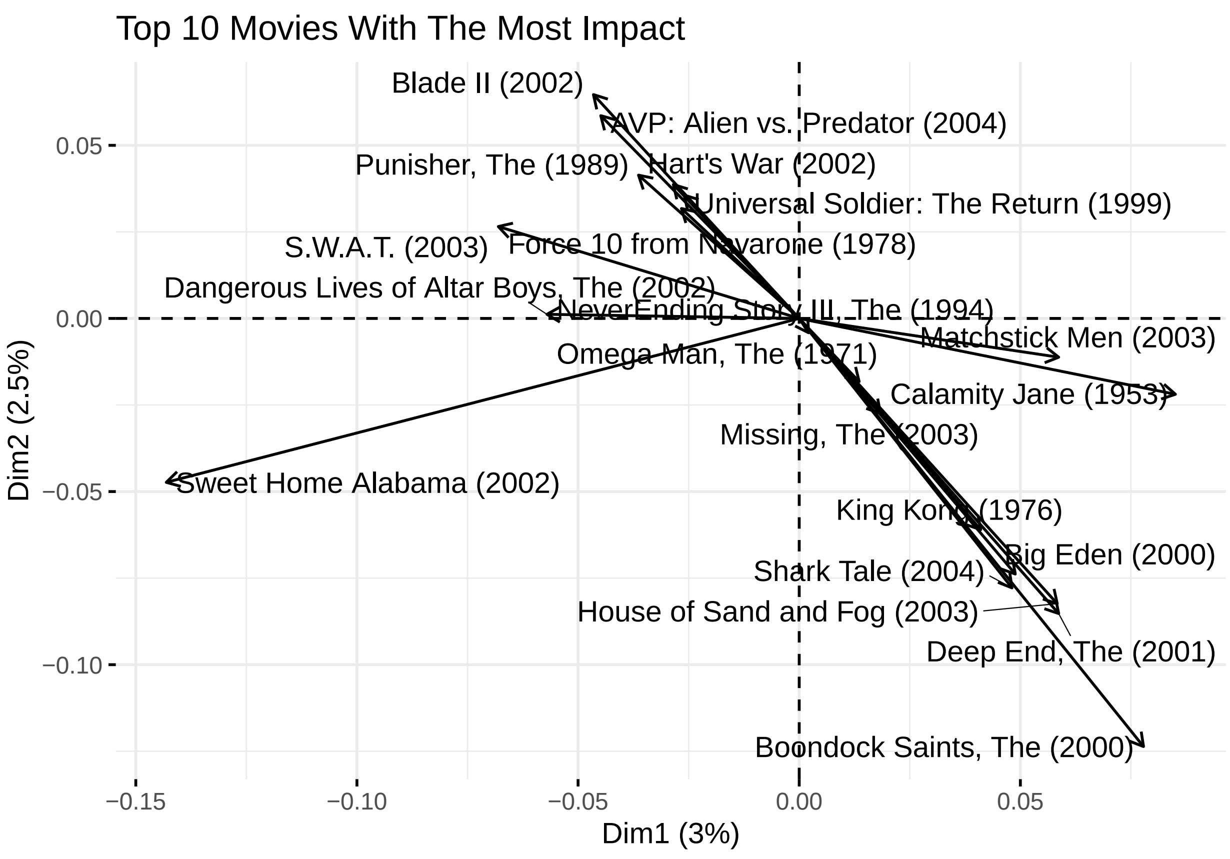

Figure 16: Dimension 1 & 2 Movie Impact

```
most_rated <- edx %>%
  group_by(movieId) %>%
  summarize(count = n(), title = first(title)) %>%
  arrange(desc(count)) %>%
  slice_head(prop = proportion)
edx <- edx %>% filter(movieId %in% most_rated$movieId)
sample_size <- nrow(edx)
cat("Top", proportion, "percent of the movies contain", sample_size*100/full_size, "percent of the rati
```

```
## Top 0.15 percent of the movies contain 79.10695 percent of the ratings.
```

```
df_list <- create_test_and_train(data_frame = edx)
train_set <- df_list$train_set
test_set <- df_list$test_set

cat("After splitting into test and train, the train set retains ", nrow(train_set)*100/full_size, "perc
```

```
## After splitting into test and train, the train set retains  63.285547 percent of the ratings.
```

4.5.3 Popular Movies PCA

While I would prefer to be able to run the PCA for the complete dataset, that is not possible. Initial tests indicate that memory requirements and run time increase exponentially with the size of the matrix used for the PCA.

Running the analysis on only the most popular movies fits within my 32 GB of memory and runs in a reasonable time. It provides good results and can be used to form the foundation of an ensemble prediction model. As a foundation model, it provides predictions in about 80% of the test cases.

```
m <- edx %>%
  group_by(movieId)%>%
  summarize(count=n()) %>%
  nrow()

u <- edx %>%
  group_by(userId)%>%
  summarize(count=n()) %>%
  nrow()

cat("The matrix of", u, "users and", m, "movies contains", m*u,
    "cells. \nSince  our popular edx subset contains", nrow(edx),
    "observations, \nthe matrix is",
    sample_size * 100/(m*u),
    "% occupied. \nThis is up from a 1.2% occupancy in the original matrix")
```

```
## The matrix of 69877 users and 1601 movies contains 111873077 cells.
## Since  our popular edx subset contains 7119669 observations,
## the matrix is 6.3640593 % occupied.
## This is up from a 1.2% occupancy in the original matrix
```

```r
# Remove unneeded objects
cleanup(keep = c("train_set", "test_set"))
```

```r
# set y_nmov = the number of movies in the training set (for later use)
y_nmov = train_set %>% group_by(movieId) %>% summarize(obs = n()) %>% nrow()
# set y_nmov = the number of movies in the training set (for later use)
y_nusers = train_set %>% group_by(userId) %>% summarize(obs = n()) %>% nrow()

#convert the dataset into a matrix with movies in columns and users in rows
#y_u,i is the entry in row u and column i => for user u and movie i
y <- train_set %>%
  select(userId, movieId, rating) %>%
  pivot_wider(names_from = movieId, values_from = rating) %>%
  as.matrix()

rm(train_set)

#add rownames and column names to facilitate exploration
rownames(y)<- y[,1]
y <- y[,-1]     #drop the userId column (the rowname now has the userId)

#save the rowMeans for reconstructing the matrix
y_row_means1 <- rowMeans(y, na.rm=TRUE)
#convert to residuals by removing the row and column averages
y <- sweep(y, 1, rowMeans(y, na.rm=TRUE))
y_col_means <- colMeans(y, na.rm=TRUE)
y <- sweep(y, 2, colMeans(y, na.rm=TRUE))

#back to the matrix developed from the actual data
y[is.na(y)] <- 0  #make the residuals with NAs = 0
y_row_means2 <- rowMeans(y, na.rm=TRUE)
y <- sweep(y, 1, rowMeans(y))

#Set a value for max principal components
my_ncp <- 10

# Run the PCA analysis
res.pca <- PCA(y, scale.unit = FALSE, graph = FALSE, ncp = my_ncp)

# Produce the forecast matrix
res.pca2 <- res.pca

# rec contains the predicted pca impact, but they are zeroed
rec <- reconst(res.pca2, ncp = my_ncp)

#This works to reconstruct the forecast from zeroed to comparable values
rec2 <- rcst(rec)

cleanup(keep = c("rec2", "test_set")) #free up memory

#calculate the predictions
pred <- as_tibble(rec2, rownames=NA) %>%
  rownames_to_column(var="userId") %>%
  pivot_longer(-userId, names_to = "movieId", values_to = "prediction") %>%
  mutate(userId = as.integer(userId), movieId = as.integer(movieId))

rm(rec2) #free up memory

# save the predictions for future use
write_rds(pred, "pca_popular.rda")

# join with test set to get ratings and predictions together
pred <- test_set %>% left_join(pred, by=c("userId","movieId"))

update_results_table(results = RMSE(pred$rating, pred$prediction),
             method = "Most Popular Movies PCA",
             type = "Test")

display_results_table("RMSE Results")
```

Table 9: RMSE Results

Method	Type	RMSE
Just the average	Test	1.0599763488047
Movie Effect Model	Test	0.943684563857829
Movie + User Effects Model	Test	0.866242706667592
Regularized Movie Effect Model	Test	0.943611139110153
Regularized Movie + User Effect Model	Test	0.865497888766143
Regularized Movie + User Effect Model: Separate Lambdas	Test	0.865497492912466
Most Popular Movies PCA	Test	0.849474113484874

Color Key
BLUE:needs work ORANGE:getting there RED:exceeds target

The RMSE values on the test set indicate that PCA produces a high quality model.

4.5.4 PCA Based Ensemble prediction

The PCA model produced above will be used as the foundation for this ensemble. Due to memory requirements to produce the PCA model, we only have predictions for about 80% of the potential cells in the matrix. We will use the regularized base model from above to fill in the blanks for the remaining 20% of the cells.

In addition to the ensemble, we are going to use a concept that I call clipping. After looking at model error, it becomes apparent that error, as represented by residuals, becomes worse the further the prediction or rating move away from the mean of 3.5. Also, an examination of the initial PCA model predictions show that there are predictions outside the range of 0.5 to 5. It is obvious that these extreme predictions are not feasible and need to be "clipped".

Clipping is essentially a form of post-processing that is done after the model runs are completed and the predictions are created. The advantage of using post-processing is that it is faster and it could be potentially used to tune multiple variables in an ensemble.

A bit of manual tuning demonstrated that the model actually performs better when it is clipped even further back from the edges. For this analysis, we will be clipping as follows:

- Any values > 4.75 will be clipped back to 4.75

- Any values < 0.75 will be clipped to 0.75

The RMSE for the ensemble prediction now covers the entire test set. The results are shown in Table 10.

```r
#Memory has been an issue here, so let's clean up and collect garbage
cleanup()
gc(verbose = FALSE)
```

```
##            used  (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells  2774337 148.2  121921466 6511.4  114832159  6132.7
## Vcells 24984945 190.7 1212419912 9250.1 1435066549 10948.7
```

```r
# load the data needed for prediction
bmp <- read_rds("base_model_predictions.rda") #from the test data set
pca_pop <- read_rds("pca_popular.rda")

# load a clean version of the training and test set
df_list <- create_test_and_train(proportion = run_size)
train_set <- df_list$train_set
test_set <- df_list$test_set

#create the ensemble
ensemble <- bmp %>%
  left_join(pca_pop, by=c("movieId", "userId")) %>%
  rename(pred_base = pred,
         pred_pca_pop = prediction)

#percent of observations with missing data from the pca popular analysis
mean(is.na(ensemble$pred_pca_pop))
```

```
## [1] 0.20897941
```

```r
#create the ensemble prediction
#the predictor is the pca result. If pca is NA, the base result is used.
ens <- ensemble %>%
  mutate(ensemble_prediction = ifelse(is.na(pred_pca_pop),pred_base, pred_pca_pop)) %>%
  right_join(test_set,by=c("movieId", "userId")) %>%
  mutate(ensemble_prediction = ifelse(ensemble_prediction > 4.8, 4.8, ensemble_prediction),
         ensemble_prediction = ifelse(ensemble_prediction < 1.0, 1.0, ensemble_prediction))

#calculate the RSME
ensemble_rsme <- RMSE(ens$rating.x, ens$ensemble_prediction)

update_results_table(results = ensemble_rsme,
             method = "PCA Ensemble Prediction",
             type = "Test")

display_results_table("RMSE Results")
```

4.6 Matrix Factorization with recosystem

When searching for a matrix factorization package to use, my priorities were efficient use of memory and speed of calculation. Typically R resides 100% in memory and is single threaded using only one core. In my

Table 10: RMSE Results

Method	Type	RMSE
Just the average	Test	1.0599763488047
Movie Effect Model	Test	0.943684563857829
Movie + User Effects Model	Test	0.866242706667592
Regularized Movie Effect Model	Test	0.943611139110153
Regularized Movie + User Effect Model	Test	0.865497888766143
Regularized Movie + User Effect Model: Separate Lambdas	Test	0.865497492912466
Most Popular Movies PCA	Test	0.849474113484874
PCA Ensemble Prediction	Test	0.835723487170772

Color Key
BLUE:needs work ORANGE:getting there RED:exceeds target

search for a multi-threaded option, I discovered recosystem. It is not only multi-threaded, but it efficiently uses both memory and disk capacity when solving.

> `recosystem` is an R wrapper of the `LIBMF` 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/), an open source library for recommender system using parallel marix factorization. (Chin, Yuan, et al. 2015)
>
> `LIBMF` is a high-performance C++ library for large scale matrix factorization. `LIBMF` itself is a parallelized library, meaning that users can take advantage of multicore CPUs to speed up the computation. It also utilizes some advanced CPU features to further improve the performance. (Chin, Yuan, et al. 2015)
>
> `recosystem` is a wrapper of `LIBMF`, hence it inherits most of the features of `LIBMF`, and additionally provides a number of user-friendly R functions to simplify data processing and model building. Also, unlike most other R packages for statistical modeling that store the whole dataset and model object in memory, `LIBMF` (and hence `recosystem`) can significantly reduce memory use, for instance the constructed model that contains information for prediction can be stored in the hard disk, and output result can also be directly written into a file rather than be kept in memory.
>
> [@Qui, Yixuan. 2020. recosystem:Recommender System Using Parallel Matrix Factorization. <https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html>]

An interesting feature of Matrix Factorization is that only the userId, movieId and rating are used to build the model.

> The idea behind matrix factorization is to capture patterns in rating data in order to learn certain characteristics, aka latent factors that describe users and items. . . . these factors are stored in two matrices, P - user factors and Q - item factors. Let's imagine that items are movies that users have rated. For movies, those factors might measure obvious dimensions such as the amount of action or comedy, orientation towards children, less well defined dimensions such as depth of character development or quirkiness; or completely uninterpretable dimensions. For users, each factor measures how much the user likes movies that score high on the corresponding movie factor." The idea behind matrix factorization is to capture patterns in rating data in order to learn certain characteristics, aka latent factors that describe users and items.
>
> [@Nikolić, Stefan. 2020. Fast matrix factorization in R. <https://blog.smartcat.io/2017/fast-matrix-factorization-in-r/>]

We are taking advantage of the multiple thread capabilities in the recosystem runs. As such, *the results are not guaranteed to be reproducible*, even if a random seed is set.[6] My experience shows only very minor variations between runs at the 4th decimal place.

4.6.1 Matrix Factorization for Movies

This is the standard approach to using matrix factorization, and will be the first model we explore. The results for this model are shown in Table 11 with and without clipping.

```r
set.seed(831, sample.kind = "Rounding")

#read in the train and test data
train_set <- read_rds("train_set.rda")
test_set <- read_rds("test_set.rda")

# Convert the train and test sets into recosystem input format
train_data <-  with(train_set, data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))
test_data  <-  with(test_set,  data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))

# Create the model object
r <-  recosystem::Reco()

# Tuning parameters were set per the recosystem vignette,
# With the exception of nthreads which will be set to the number of cores
cores <- parallel::detectCores(all.tests = FALSE, logical = TRUE)
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread  = cores,
                                       niter = 10))
# Tuning is a time-consuming step, but produce a reusable set of options
# Save the options for future use
write_rds(opts, "recosys_opts_movies.rda")

# Train the algorithm
r$train(train_data, opts = c(opts$min,
                             nthread = cores,
                             niter = 50,
                             verbose = FALSE))

# Calculate the predicted values
pred <-  r$predict(test_data, out_memory())

update_results_table(results = RMSE(test_set$rating, pred),
             method = "Matrix Factorization",
             type = "Test")
```

[6] Qiu, Yixuan. 2016. recosystem: Recommender System Using Parallel Matrix Factorization. https://statr.me/2016/07/recommender-system-using-parallel-matrix-factorization/

```r
# Calculate results after clipping extreme predictions back
clip <- function(prediction){
  if (prediction > 4.75){
    prediction <- 4.75
  } else if (prediction < 0.75){
    prediction <- 0.75
  }
  return(prediction)
}

pred <- sapply(pred, clip)

update_results_table(results = RMSE(test_set$rating, pred),
             method = "Matrix Factorization with Clipping",
             type = "Test")

display_results_table("RMSE Results")
```

Table 11: RMSE Results

Method	Type	RMSE
Just the average	Test	1.0599763488047
Movie Effect Model	Test	0.943684563857829
Movie + User Effects Model	Test	0.866242706667592
Regularized Movie Effect Model	Test	0.943611139110153
Regularized Movie + User Effect Model	Test	0.865497888766143
Regularized Movie + User Effect Model: Separate Lambdas	Test	0.865497492912466
Most Popular Movies PCA	Test	0.849474113484874
PCA Ensemble Prediction	Test	0.835723487170772
Matrix Factorization	Test	0.790704460790676
Matrix Factorization with Clipping	Test	0.789938918638324

Color Key
BLUE:needs work ORANGE:getting there RED:exceeds target

4.6.2 Matrix Factorization for Genres

The standard matrix factorization model uses three variables, user ID, movie ID and rating. In an effort to see what the impact is for genres, we will use the genre factor level instead of the movie ID. We are using the same prediction model that we used above for movies, but this time we will pick the user ratings by combined genres. The results of this run are shown in Table 12

```r
#Run recosystems and this time recommend genres instead of movies

cleanup()

#For the recosystem model to work, we need to convert genres to factors
#so that they can be treated as an integer numeric. We will perform that
#conversion on the edx data frame before sending it to the test and train
#create function. This will ensure that the factor levels are consistent
```

```r
#on the test and train data frames
edx_factors <- read_rds("edx.rda") %>%
  mutate(genres = as.factor(genres))

df_list <- create_test_and_train(proportion = run_size,
                                 data_frame = edx_factors)
train_set <- df_list$train_set
test_set <- df_list$test_set

set.seed(831, sample.kind = "Rounding")

# Convert the train and test sets into recosystem input format
train_data <-  with(train_set, data_memory(user_index = userId,
                                           item_index = genres,
                                           rating     = rating))
test_data  <-  with(test_set,  data_memory(user_index = userId,
                                           item_index = genres,
                                           rating     = rating))

# Create the model object
r <-  recosystem::Reco()

# Tuning parameters were set per the recosystem vignette,
# With the exception of nthreads which will be set to the number of cores
cores <- parallel::detectCores(all.tests = FALSE, logical = TRUE)
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread  = cores,
                                       niter = 10))

# Tuning is a time-consuming step, but produce a reusable set of options
# Save the options for future use
write_rds(opts, "recosys_opts_genres.rda")

# Train the algorithm
r$train(train_data, opts = c(opts$min,
                             nthread = cores,
                             niter = 50,
                             verbose=FALSE))

# Calculate the predicted values
pred <-  r$predict(test_data, out_memory())

update_results_table(results = RMSE(test_set$rating, pred),
             method = "Matrix Factorization of Genres",
             type = "Test")

display_results_table("RMSE Results - Full Test Suite")
```

Table 12: RMSE Results - Full Test Suite

Method	Type	RMSE
Just the average	Test	1.0599763488047
Movie Effect Model	Test	0.943684563857829
Movie + User Effects Model	Test	0.866242706667592
Regularized Movie Effect Model	Test	0.943611139110153
Regularized Movie + User Effect Model	Test	0.865497888766143
Regularized Movie + User Effect Model: Separate Lambdas	Test	0.865497492912466
Most Popular Movies PCA	Test	0.849474113484874
PCA Ensemble Prediction	Test	0.835723487170772
Matrix Factorization	Test	0.790704460790676
Matrix Factorization with Clipping	Test	0.789938918638324
Matrix Factorization of Genres	Test	0.911545518858134

Color Key
BLUE:needs work ORANGE:getting there RED:exceeds target

4.6.3 Matrix Factorization Ensemble

Although the results for the MF Genres model was significantly worse than the MF Movies Model, I decided to see if an ensemble approach might still offer improvement. The ensemble used is weighted 94% to the MF Movie model and also includes clipping. As shown in Table 13, there is a small but measurable improvement.

```
# cleanup memory
cleanup()

# read in the edx data frame and convert genres to a factor
edx_factors <- read_rds("edx.rda") %>%
  mutate(genres = as.factor(genres))

# create the test and train sets
df_list <- create_test_and_train(proportion = run_size,
                                 data_frame = edx_factors)
train_set <- df_list$train_set
test_set <- df_list$test_set

set.seed(831, sample.kind = "Rounding")

# Convert the train and test sets into recosystem input format
train_data <-  with(train_set, data_memory(user_index = userId,
                                            item_index = movieId,
                                            rating     = rating))
test_data  <-  with(test_set,  data_memory(user_index = userId,
                                            item_index = movieId,
                                            rating     = rating))

########################################################
# Recreate the MF Movies Model
cores <- parallel::detectCores(all.tests = FALSE, logical = TRUE)

#load the tuning parameters
opts <- read_rds("recosys_opts_movies.rda")
```

```r
#create the recosystem modeling object
r <-  Reco()

#train the model
r$train(train_data, opts = c(opts$min,
                             nthread = cores,
                             niter = 50,
                             verbose=FALSE))

# Calculate the predicted values for movies
pred <-  r$predict(test_data, out_memory())

########################################################
# Recreate the MF Genres Model

# load the saved tuning options
opts <- read_rds("recosys_opts_genres.rda")

# Convert the train and test sets into recosystem input format
train_data <-  with(train_set, data_memory(user_index = userId,
                                           item_index = genres,
                                           rating     = rating))
test_data  <-  with(test_set,  data_memory(user_index = userId,
                                           item_index = genres,
                                           rating     = rating))
# train the model
r$train(train_data, opts = c(opts$min,
                             nthread = cores,
                             niter = 50,
                             verbose=FALSE))

# Calculate the predicted values
pred2 <-  r$predict(test_data, out_memory())

# Create the Matrix Factorization Ensemble
ensemble_mf <- tibble(movies = pred, genres = pred2)
# add clipping parameters to the ensemble
ens <- ensemble_mf %>%
  mutate(prediction = (movies * 0.94) + (genres * 0.06)) %>%
  mutate(prediction = ifelse(prediction > 4.75, 4.75, prediction),
         prediction = ifelse(prediction < 0.75, 0.75, prediction))

#calculate the rmse results
ensemble_rsme <- RMSE(test_set$rating, ens$prediction)

update_results_table(ensemble_rsme,
                     method = "Matrix Factorization Ensemble",
                     type = "Test")
display_results_table("Final Results")
```

Table 13: Final Results

Method	Type	RMSE
Just the average	Test	1.0599763488047
Movie Effect Model	Test	0.943684563857829
Movie + User Effects Model	Test	0.866242706667592
Regularized Movie Effect Model	Test	0.943611139110153
Regularized Movie + User Effect Model	Test	0.865497888766143
Regularized Movie + User Effect Model: Separate Lambdas	Test	0.865497492912466
Most Popular Movies PCA	Test	0.849474113484874
PCA Ensemble Prediction	Test	0.835723487170772
Matrix Factorization	Test	0.790704460790676
Matrix Factorization with Clipping	Test	0.789938918638324
Matrix Factorization of Genres	Test	0.911545518858134
Matrix Factorization Ensemble	Test	0.789821422161256

Color Key
BLUE:needs work ORANGE:getting there RED:exceeds target

5 Results

5.1 Model Validation

Model Validation will be performed in four steps.

1. PCA Ensemble Model with the combined PCA and Base model predictions
2. Matrix Factorization Movies Model
3. Matrix Factorization Genres Model
4. Matrix Factorization Ensemble

All validation results will be displayed in Table 14 at the end of this section.

```r
# run against the validation set
validation <- read_rds("validation.rda")
# Use the entire edx data set to train the model
# Run results have demonstrated that the larger data sets yield better results
train_set <- read_rds("edx.rda")
# input the results from the pca most popular movie analysis
pca_pop <- read_rds("pca_popular.rda")

# l holds the user lambda
l <- 5.1
# l_m holds the movie lambda
l_m <- 4.7
mu <- mean(train_set$rating)
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l_m))
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+1))
best_predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u, residual = rating-pred)
ensemble <- best_predicted_ratings %>%
  left_join(pca_pop, by=c("movieId", "userId")) %>%
  rename(pred_base = pred,
         pred_pca_pop = prediction)

# create the ensemble prediction
# the predictor is the pca result. If pca is NA, the base result is used.
# we will also add clipping logic to the ensemble prediction
ens <- ensemble %>%
  mutate(ensemble_prediction = ifelse(is.na(pred_pca_pop),pred_base, pred_pca_pop)) %>%
  right_join(validation,by=c("movieId", "userId")) %>%
  mutate(ensemble_prediction = ifelse(ensemble_prediction > 4.75, 4.75, ensemble_prediction),
         ensemble_prediction = ifelse(ensemble_prediction < 0.75, 0.75, ensemble_prediction))

ensemble_rsme <- RMSE(ens$rating.x, ens$ensemble_prediction)

update_results_table(results = ensemble_rsme,
                     method = "PCA Ensemble Prediction",
                     type = "Validation")

cleanup(keep = "validation")
```

```
#recosystem movies final validation
set.seed(831, sample.kind = "Rounding")

# Use the entire edx data set to train the model
# Convert 'edx' and 'validation' sets to recosystem input format
edx <- read_rds("edx.rda")
edx_reco <-  with(edx, data_memory(user_index = userId,
                                   item_index = movieId,
                                   rating = rating))

#convert validation to recosystem input format
validation_reco  <-  with(validation, data_memory(user_index = userId,
                                                  item_index = movieId,
                                                  rating = rating))

# Create the model object
r <-  Reco()
cores <- parallel::detectCores(all.tests = FALSE, logical = TRUE)

# read in the tuning parameters from the test run
opts <- read_rds("recosys_opts_movies.rda")

#train the model
r$train(edx_reco, opts = c(opts$min,
                           nthread = cores,
```

```r
                           niter = 50,
                           verbose = FALSE))

#create the predictors
pred <-  r$predict(validation_reco, out_memory())

update_results_table(results = RMSE(validation$rating, pred),
                     method = "Matrix Factorization",
                     type = "Validation")
```

```r
#recosystem genres final validation

set.seed(831, sample.kind = "Rounding")

# Use the entire edx data set to train the model
# Convert genres to a factor for use in this model
# Convert 'edx' and 'validation' sets to recosystem input format
edx <- read_rds("edx.rda") %>%
  mutate(genres = as.factor(genres))
edx_reco <-  with(edx, data_memory(user_index = userId,
                                   item_index = genres,
                                   rating = rating))
# Convert genres to a factor for use in this model
# Make sure factor levels in the validation data frame match with the edx df
# First we will create a data frame that can be used as a cross reference
# to the edx genres levels
genres_xref <- edx %>%
  mutate(genres_level = as.numeric(genres)) %>%
  group_by(genres_level) %>%
  summarize(genres = first(genres),
            genres_level = first(genres_level))
# Then we will join this to the validation data frame
validation <- validation %>%
  left_join(genres_xref, by="genres")
# Create the validation recosystem object using the genres_level column
validation_reco  <-  with(validation, data_memory(user_index = userId,
                                                  item_index = genres_level,
                                                  rating = rating))

# Create the model object
r <-  Reco()
cores <- parallel::detectCores(all.tests = FALSE, logical = TRUE)

# read in the tuning parameters from the test run
opts <- read_rds("recosys_opts_genres.rda")

#train the model
r$train(edx_reco, opts = c(opts$min,
                           nthread = cores,
                           niter = 50,
                           verbose = FALSE))

#create the predictors
```

```
pred2 <-  r$predict(validation_reco, out_memory())

update_results_table(results = RMSE(validation$rating, pred2),
                     method = "Matrix Factorization Genres",
                     type = "Validation")
```

```
#Create the Matrix Factorization Ensemble
ensemble_mf <- tibble(movies = pred, genres = pred2)

ens <- ensemble_mf %>%
  mutate(prediction = (movies * 0.94) + (genres * 0.06)) %>%
  mutate(prediction = ifelse(prediction > 4.75, 4.75, prediction),
         prediction = ifelse(prediction < 0.75, 0.75, prediction))

ensemble_rsme <- RMSE(validation$rating, ens$prediction)

update_results_table(ensemble_rsme,
                     method = "Matrix Factorization Ensemble",
                     type = "Validation")
display_results_table("Final Results")
```

Table 14: Final Results

Method	Type	RMSE
Just the average	Test	1.0599763488047
Movie Effect Model	Test	0.943684563857829
Movie + User Effects Model	Test	0.866242706667592
Regularized Movie Effect Model	Test	0.943611139110153
Regularized Movie + User Effect Model	Test	0.865497888766143
Regularized Movie + User Effect Model: Separate Lambdas	Test	0.865497492912466
Most Popular Movies PCA	Test	0.849474113484874
PCA Ensemble Prediction	Test	0.835723487170772
Matrix Factorization	Test	0.790704460790676
Matrix Factorization with Clipping	Test	0.789938918638324
Matrix Factorization of Genres	Test	0.911545518858134
Matrix Factorization Ensemble	Test	0.789821422161256
PCA Ensemble Prediction	Validation	0.854178411184051
Matrix Factorization	Validation	0.780540501782946
Matrix Factorization Genres	Validation	0.906334815480591
Matrix Factorization Ensemble	Validation	0.77984129605773

Color Key
BLUE:needs work ORANGE:getting there RED:exceeds target

5.2 Results Recap

The final results are shown in Table 14. The RMSE results shown in red are values that exceed the class target objective of an RMSE below 0.86490. Both the PCA Ensemble and the Matrix Factorization Ensemble achieved the desired result. In addition, the Matrix Factorization Movie Model provided excellent standalone results. Clipping enhanced the results on all models where it was tested. Matrix Factorization Ensemble achieved the best result with an RSME of 0.78. I am rounding this value to 0.78 because of the lack of repeatability in the model. As noted previously, due to the multi-threading architecture, repeatability is not guaranteed, even when a seed value is set.

As expected, the PCA Ensemble model performed slightly worse against the validation data set than it did against the test set. On the other hand, the MF models all performed better against the validation set. My hypothesis is that this is because the tuning parameters were developed and tested using the edx data split 80/20 into a test and training set. These tuning parameters were used for the final validation model, but the training of the model occurred on the full edx data set. My test runs showed that the MF model performance improved significantly as the data used for training the model increased. I believe that the 20% increase in training data achieved by using the entire edx dataset improved the model performance.

6 Conclusion

The base model is extremely helpful as a starting tool. I believe that it is very understandable and provides a great starting point for exploring machine learning and for understanding the original Bellkor approach to the recommendation problem. It has the advantage of being easy on memory and the quickest model to run. It's disadvantages are that it requires quite a bit of analyst insight to extend it, and that even the best case from the Bellkor efforts cannot compete with Matrix Factorization (MF).

From the perspective of producing a predictor model, Matrix Factorization was the clear winner. It produced significantly better RMSE results, was a simpler model to implement, ran much faster, and was less memory constrained due to the construction of the recosystems package.

Part of the reason for creating the Genres focused version of the Matrix Factorization was to see how much of the genres signal was lost in the original MF model. The results demonstrate that only a slight improvement was possible by adding a genres model in an ensemble. For a future recommender system project, the Movies MF model would be my solution choice.

The PCA analysis would probably perform better on a machine with larger memory where more than a 15% sample of movies could be used. The FactoMineR package also provides significantly better data visualization and analysis tools. If my goal were to further understand the data and "why" things were happening in the model, PCA would be my choice.

Post-processing was an interesting discovery. When looking at the errors in the predictions, it was obvious that they were occurring at the extreme predictions of the model. Post-processing via clipping provided measurable improvement. This also allows clipping to be separated from large model runs and applied and manipulated at the ensemble level.

This project was bounded in scope by time. There were many more permutations and combinations of modeling approaches that could be attempted. The project was also bounded by the hardware restrictions of a home desktop computer. More time and better hardware would surely lead to improvements.

Other potential options for improving this solution include but are not limited to:

- Base Model - extend with genres and time stamp components
- PCA Analysis - run the full model and add a genres version of the solution to the ensemble
- Matrix Factorization - run in an ensemble that also incorporated the Base Model and the PCA Analysis. Check the impact of training the model on datasets of different sizes.
- There is opportunity for other sensitivity analysis including:
  - evaluating the impact of different seed values on the predicted RMSE
  - tuning the clipping values
- There are additional packages and modeling approaches that could be evaluated. A good example of a package to evaluate would be recommenderlab. There are others in the CRAN library.

- Improve the modeling of the genre variable by incorporating effect-modeling and zeroing.

In conclusion, this project is in no way an “optimal solution”, but it does provide very good results and has been an excellent learning tool.

7 References

Bell, Robert, Yehuda Koren and Chris Volinsky. 2008. The BellKor solution to the Netflix Prize

Gailloty, Cleris. 2018. Comprehensive PCA with R using FactoMineR. https://www.kaggle.com/agailloty/comprehensive-pca-with-r-using-factominer

Husson, Francois, Julie Josse, Sebastien Le, Jeremy Mazet. 2020. FactoMineR: Multivariate Exploratory Data Analysis and Data Mining. https://CRAN.R-project.org/package=FactoMineR

Kassambara, Alboukadel. 2017. Principal Component Methods in R: Practical Guide. http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/118-principal-component-analysis-in-r-prcomp-vs-princomp/

Irizzary, Rafael, 2019. Introduction to Data Science. https://leanpub.com/datasciencebook

Nikolić, Stefan. 2020. Fast matrix factorization in R. https://blog.smartcat.io/2017/fast-matrix-factorization-in-r/

Qiu, Yixuan. 2016. recosystem: Recommender System Using Parallel Matrix Factorization. https://statr.me/2016/07/recommender-system-using-parallel-matrix-factorization/

Qiu, Yixuan. 2020. recosystem: Recommender System Using Parallel Matrix Factorization. https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html