Capstone Project 1: Reccomender

Sagar Apshankar

5/13/2021

Introduction

Abstract

Recommendation systems seek to predict the preference a user would give to an item based on other preferences the user has expressed and the preferences of other users pertaining to the item in question. Recommendation systems are one of the most ubiquitous applications of Machine Learning. They are used by e-marketplaces such as Amazon and streaming platforms such as Netflix. In fact, in 2006, Netflix offered a prize of 1 million USD to whoever that could increase the accuracy of their recommendation engine by 10%. In this assignment, we evaluate different algorithms to try and predict the rating of users for movies. We evaluate our prediction by minimizing Root Mean Squared Error(RMSE) between predicted and real ratings from a test set. The best performing model of those evaluated was recosystem Matrix Factorization algorithm followed by Linear Regression with Regularized Movie, User and Genre effects. The respective RMSEs were 0.858 and 0.865.

Executive Summary

In this project we use the 10M version of the MovieLens dataset generated by the GroupLens research lab. The data is divided at the outset into training, test and hold-out validation sets.

In this report, we will complete the following steps: +Examine the data for finding which factors may affect the rating +Wrangle the data so as to be workable with existing recommender engines +We will train recommendation algorithms using the training set and compare them by evaluating the RMSE with the test set. +In the conclusion,we will slect the best model and report the RMSE obtained by evaluating our algorithm on the hold-out validation set.

Dataset

Data Acquisition and dependencies

The following libraries are needed at different points in the analysis.

```
packages_required <-c("dplyr", # Wrangling
  "ggplot2", #for graphics
  "stringr", #string operations
  "tidyr", # Wrangling
  "caret", #for CreateDataPartition and train()
  "ggthemes", #for graphics themes
  "lubridate", #for dealing with dates
  "tidyverse", # Wrangling
  "data.table", #for wrangling
  "Matrix", #for SparseMatrix
  "recosystem", #for Matrix Factorization model
  "ggthemes", #for ggplot theme economist
  "recommenderlab",#for UBCF,IBCF, POPULAR models
  "knitr" #for RMD options
  )

using<-function(...) {
  libs<-unlist(list(...))
  req<-unlist(lapply(libs,require,character.only=TRUE))
  need<-libs[req==FALSE]
  if(length(need)>0){
    install.packages(need)
    lapply(need,require,character.only=TRUE)
  }
}


using(packages_required)


knitr::opts_chunk$set(cache = TRUE, warning = FALSE,
                      message = FALSE, cache.lazy = FALSE)

options(tinytex.verbose = TRUE)
```

The following code downloads the data, partitions it deterministically into test, train and validation sets and ensures that the sets don't contain any users or movies exclusively.

```
##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

knitr::opts_chunk$set(cache = TRUE) #Don't re-run code unless it has changed

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

# if using R 3.6 or earlier:
#movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
#                                            title = as.character(title),
#                                           genres = as.character(genres))
# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                            title = as.character(title),
                                            genres = as.character(genres))


movielens <- left_join(ratings, movies, by = "movieId")


# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

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

rm(dl, ratings, movies, temp, movielens, removed,test_index)
```

Data Exploration

Let us explore the dataframes created to identify the factors which make up a movie rating.

```
paste("The dataset has " , nrow(edx) , "ratings")
```

Basic Characteristics

```
## [1] "The dataset has  9000055 ratings"
```

```
paste("There are " , ncol(edx), "columns")
```

```
## [1] "There are  6 columns"
```

```
#Checking data validity

paste("There are " , edx %>% filter(rating==0) %>% nrow(), "ratings = 0")
```

```
## [1] "There are  0 ratings = 0"
```

```
paste("There are " , edx %>% filter(rating>5) %>% nrow(), "ratings > 5")
```

```
## [1] "There are  0 ratings > 5"
```

```
paste("There are " , edx %>% distinct(movieId) %>% count(), "distinct movies")
```

```
## [1] "There are  10677 distinct movies"
```

```
paste("There are " ,edx %>% distinct(userId) %>% count(), "distinct users")
```

```
## [1] "There are  69878 distinct users"
```

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
glimpse(edx)
```

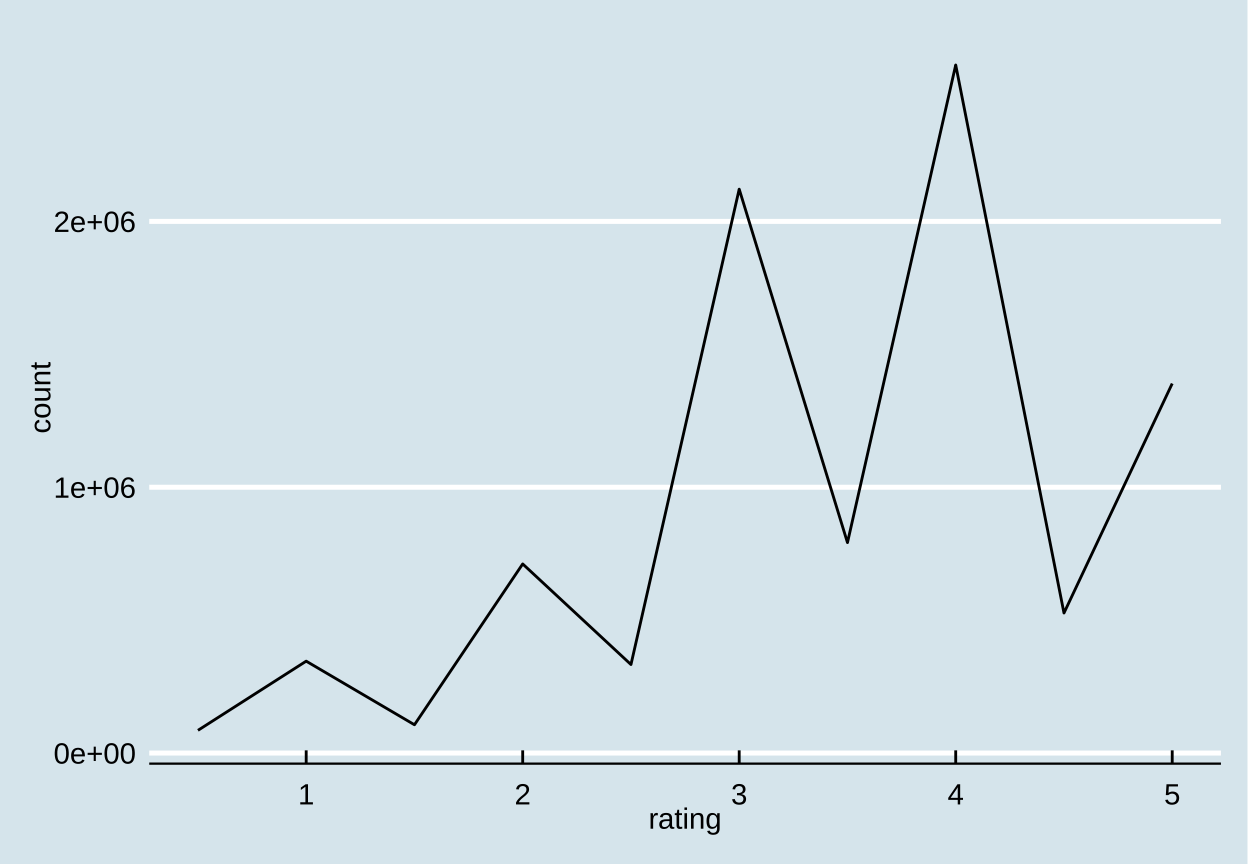
```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

We see that the dataframe provided has 9000055 rows and 6 columns. The ratings are between 0 and 5 with missing values represented by 0. There are 10677 distinct movies and 69878 distinct users.

From the summary, we have the mean rating at 3.512 and the median at 4.

From glimpse, we understand that variability in the rating may come from any combination of the user, movie, timestamp and genre.

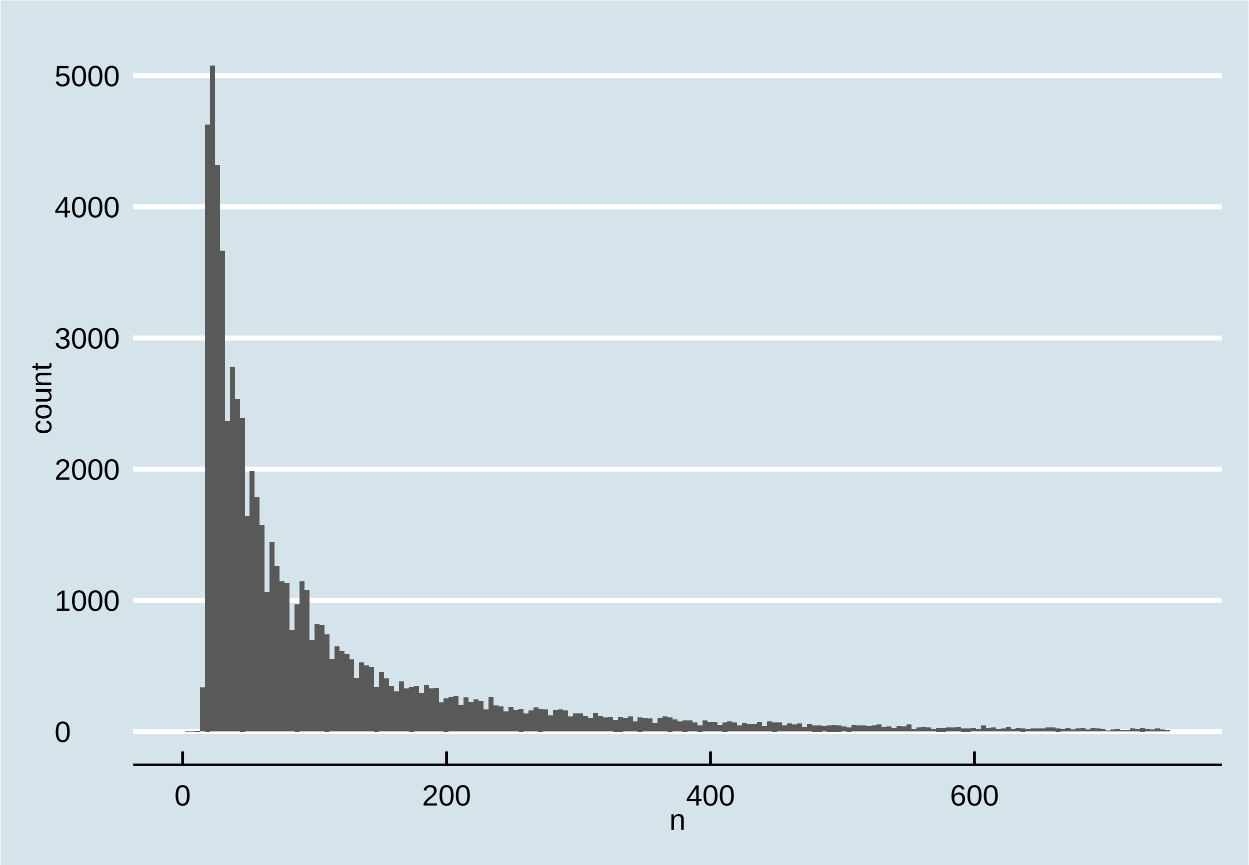
Effects and distribution Let us see the distribution of the ratings.



The plot is consistent with our observations about the median. We observe that half star ratings are less common than whole number ratings.

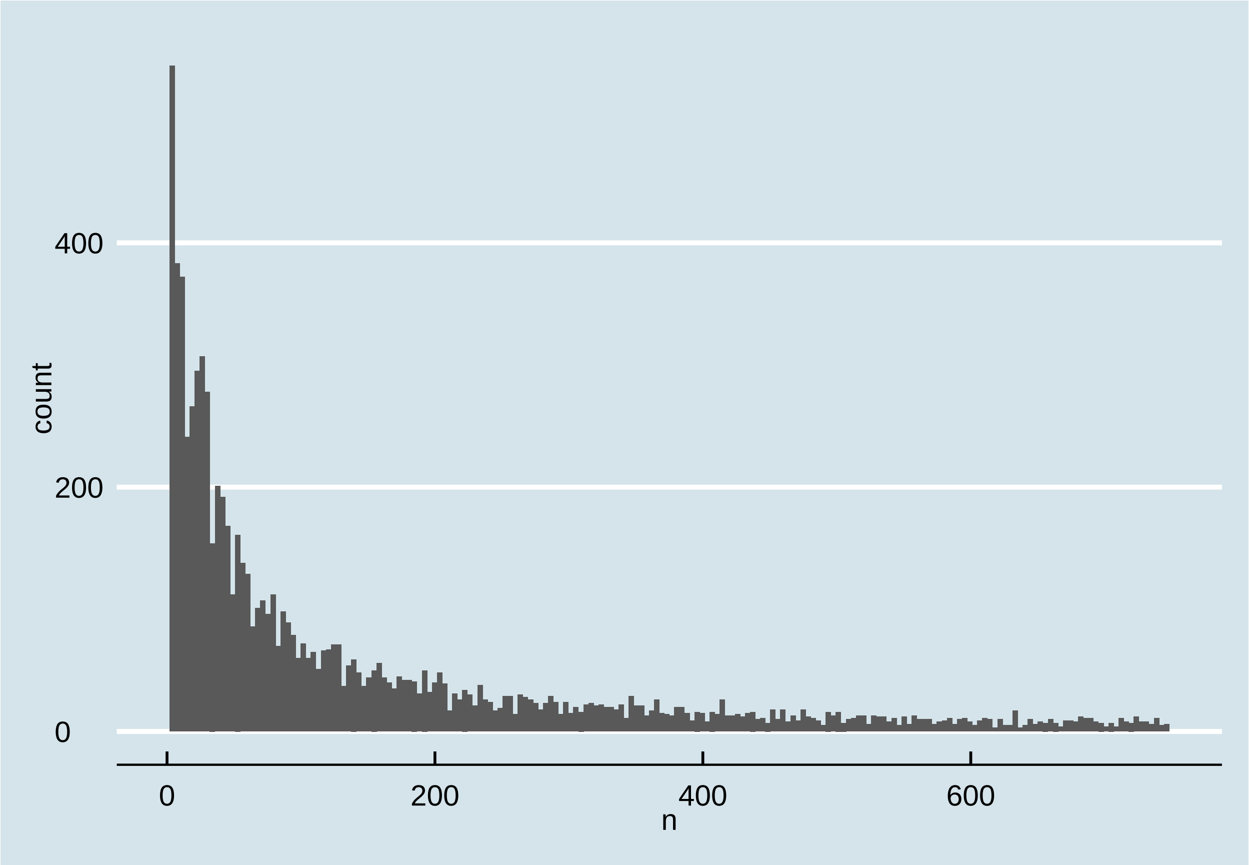
Let us examine the manner in which different users rate movies.

```
edx %>%
  group_by(userId)%>%
  summarise(n=n())%>%
  arrange(desc(n))%>%
  ggplot(aes(n))+
  geom_histogram(bins=200)+
  scale_x_continuous(limits = c(0,750))+
  theme_economist()
```



The above graph shows that most users give few ratings and that there are a few who give very high number of ratings. This means we may be able to sample the data to reduce computation and still get a workable model.

```
edx %>%
  group_by(movieId)%>%
  summarise(n=n())%>%
  arrange(desc(n))%>%
  ggplot(aes(n))+
  geom_histogram(bins=200)+
  scale_x_continuous(limits = c(0,750))+
  theme_economist()
```

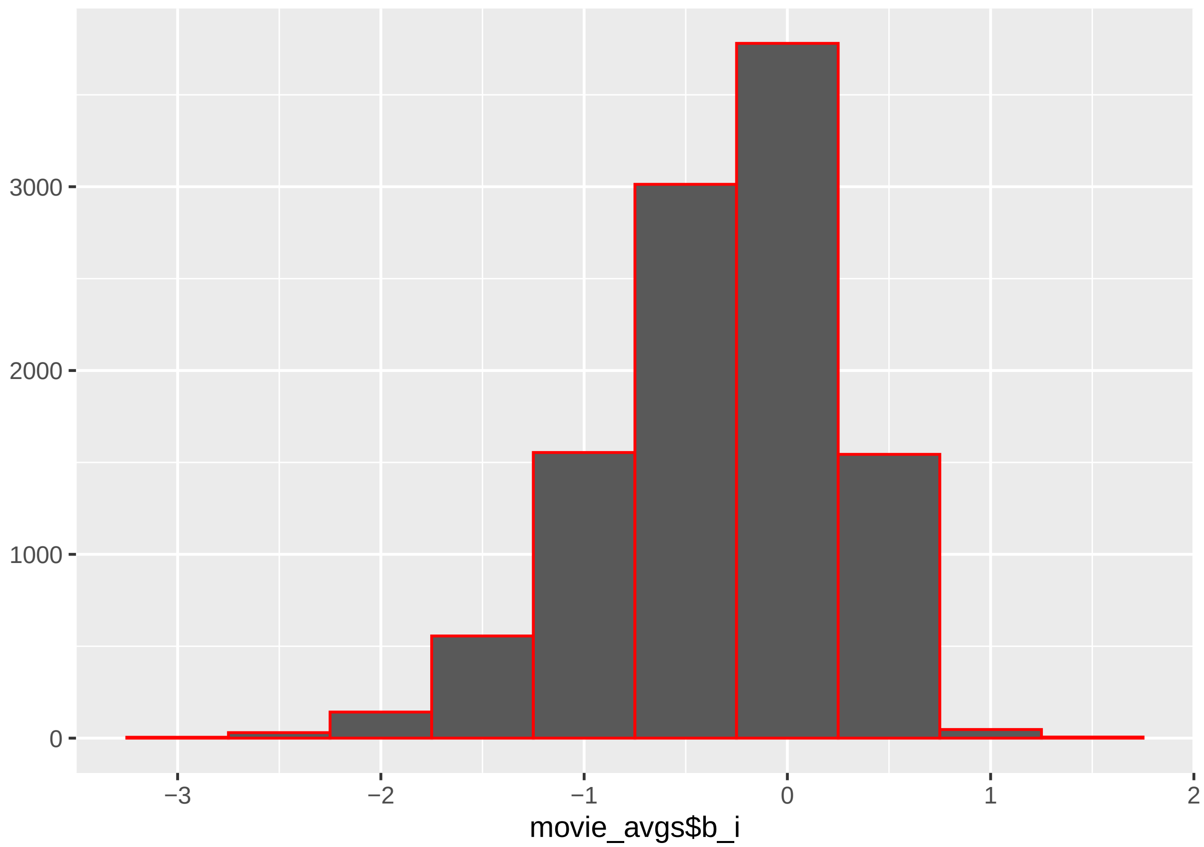


Similarly, we can observe that most movies receive very few ratings and few receive lots. These will be the ones which will help us make our model accurate.

Let us now examine if some users have a propensity to rate movies better than others and if some movies have a tendency to be rated better than others.

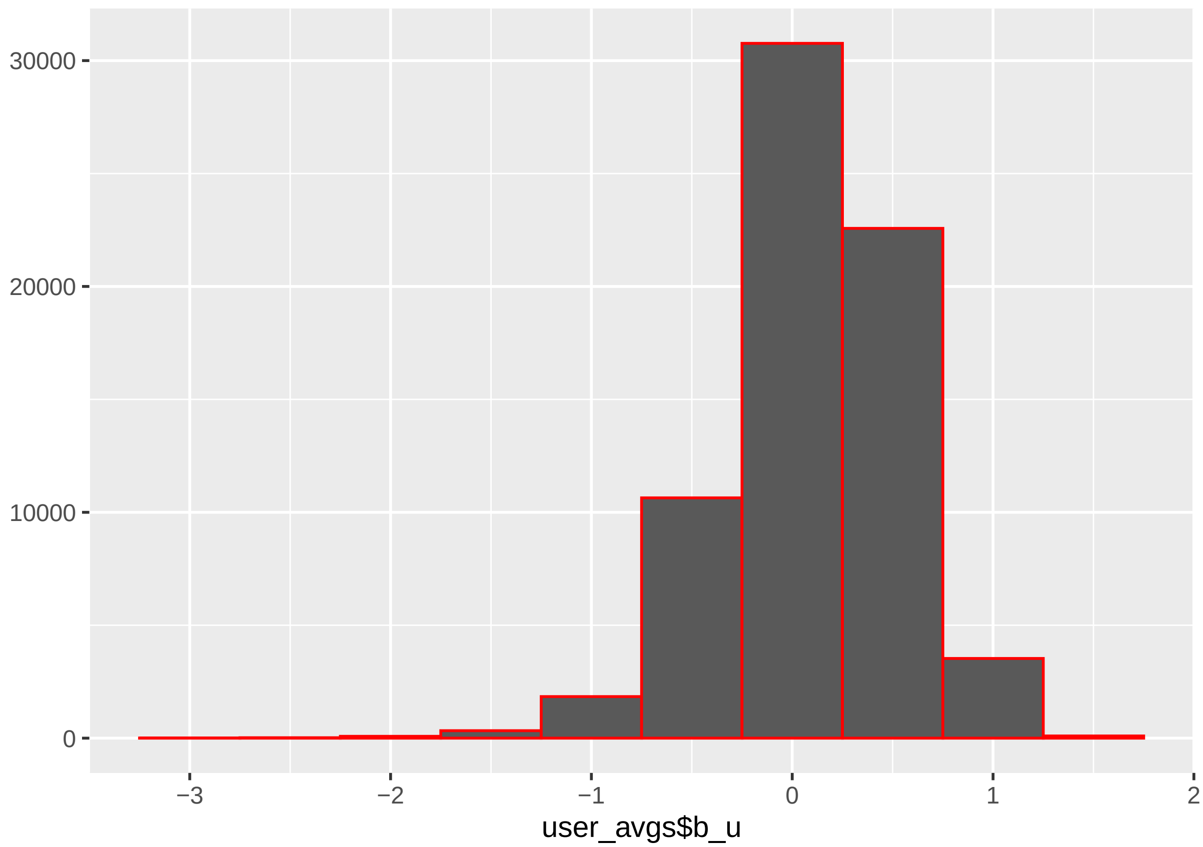
In order to do this, we will first subtract the mean rating from all ratings, then the movie mean and finally the user mean to ensure that we see only the variability attributed to the particular factor.

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

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

qplot(movie_avgs$b_i,bins=10,color=I("red"))
```



From this plot, it is clear that there is a lot of variability in the way movies are rated. There exists a "movie effect"., in other words, each movie is rated differently.

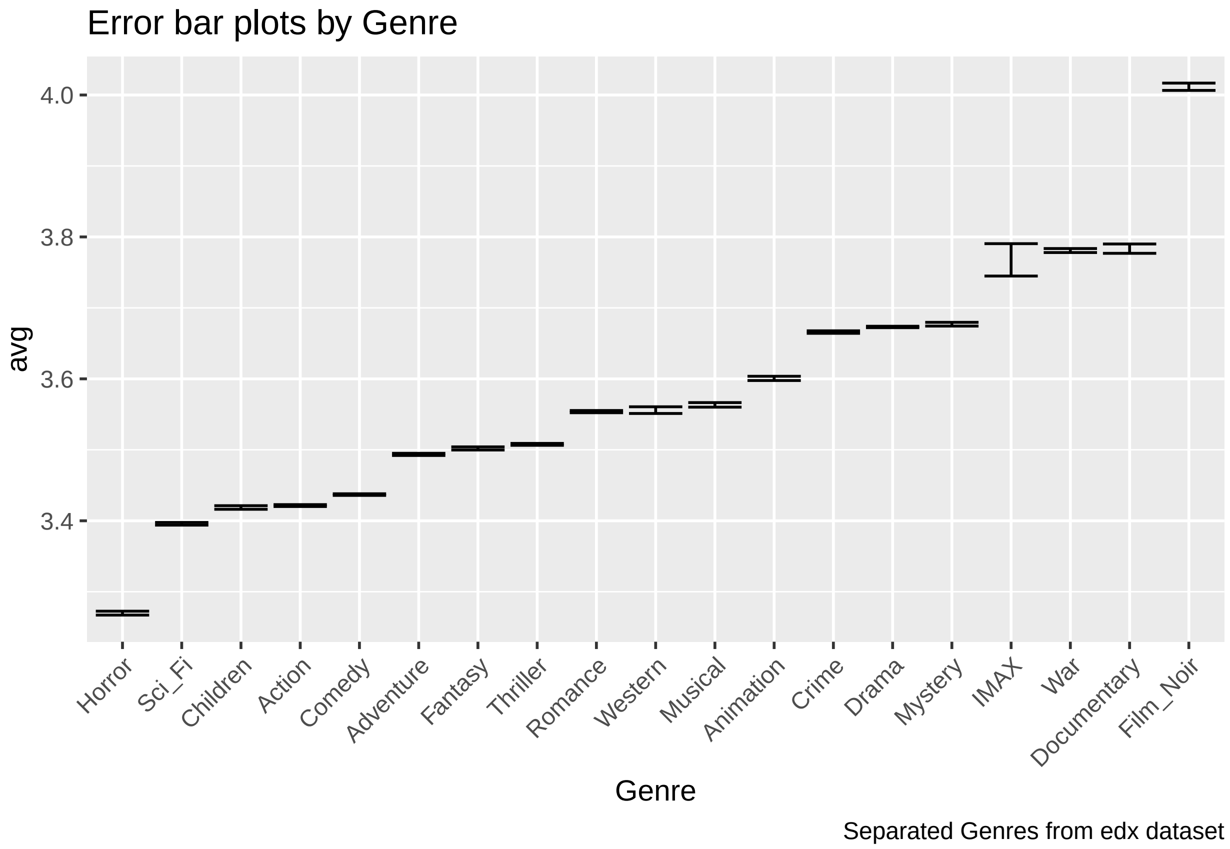
```
mu <- mean(edx$rating)

user_avgs <- edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu))

qplot(user_avgs$b_u,bins=10,color=I("red"))
```



We see from the histogram that more users are easily pleased because there are more individuals who give a rating above the mean, but there do exist some hard-to-please or cranky users. Isolating the user effect is thus a good idea.

We will now examine the genres to see if certain genres are rated better than others. Since the genres are grouped and joined, we will need to encode them into separate columns first.

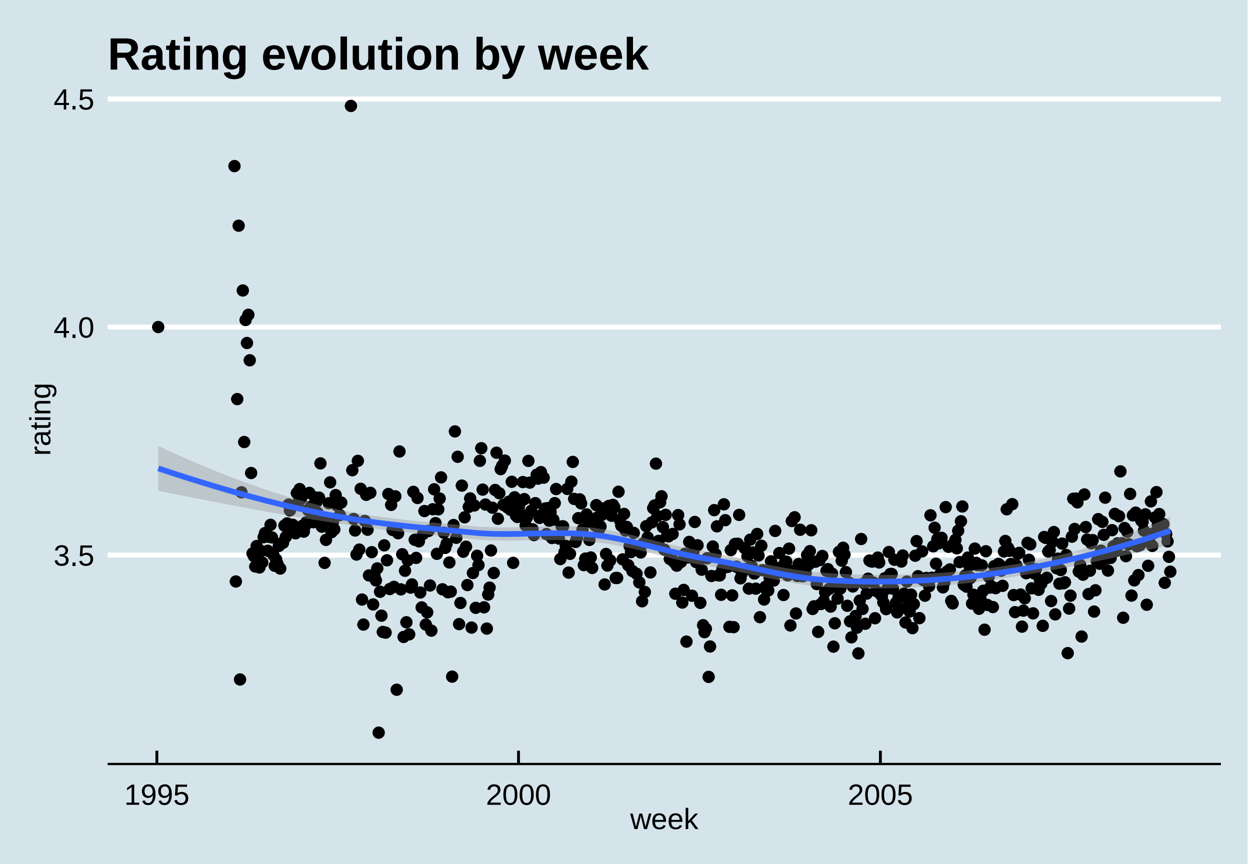
```
knitr::opts_chunk$set(cache = TRUE)
#The following code is decodes the genre data by adding a 0 or an NA based on
#whether the name of the genre is present in the genre column
n_a=NA
genre_data <-   edx %>% mutate(
  Romance = ifelse(str_detect(genres,"Romance"),1*rating,n_a),
  Comedy = ifelse(str_detect(genres,"Comedy"),1*rating,n_a),
  Action = ifelse(str_detect(genres,"Action"),1*rating,n_a),
  Crime = ifelse(str_detect(genres,"Crime"),1*rating,n_a),
  Thriller = ifelse(str_detect(genres,"Thriller"),1*rating,n_a),
  Drama = ifelse(str_detect(genres,"Drama"),1*rating,n_a),
  Sci_Fi = ifelse(str_detect(genres,"Sci-Fi"),1*rating,n_a),
  Adventure = ifelse(str_detect(genres,"Adventure"),1*rating,n_a),
  Children = ifelse(str_detect(genres,"Children"),1*rating,n_a),
  Fantasy = ifelse(str_detect(genres,"Fantasy"),1*rating,n_a),
  War = ifelse(str_detect(genres,"War"),1*rating,n_a),
  Animation = ifelse(str_detect(genres,"Animation"),1*rating,n_a),
  Musical = ifelse(str_detect(genres,"Musical"),1*rating,n_a),
  Western = ifelse(str_detect(genres,"Western"),1*rating,n_a),
  Mystery = ifelse(str_detect(genres,"Mystery"),1*rating,n_a),
  Film_Noir = ifelse(str_detect(genres,"Film-Noir"),1*rating,n_a),
  Horror = ifelse(str_detect(genres,"Horror"),1*rating,n_a),
  Documentary = ifelse(str_detect(genres,"Documentary"),1*rating,n_a),
  IMAX = ifelse(str_detect(genres,"IMAX"),1*rating,n_a)
)


#In order to make a errorbar plot , we now need a table containing genre and rating
# we first gather all the columns into a single values column called rating and then
# we filter out all the na rows
# later we calculate the 2se for making the error plot
genre_data %>%
  select(Romance:IMAX) %>%
  gather(., key="Genre",value = "rating") %>%
  filter(is.na(rating)==FALSE) %>%
  group_by(Genre) %>%
  summarise(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n)) %>%
  mutate(Genre = reorder(Genre, avg)) %>%
  ggplot(aes(x = Genre, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Error bar plots by Genre" , caption = "Separated Genres from edx dataset")
```



Clearly, the genre effect is quite pronounced and we will choose it as a part of our model.

The last aspect to check is that of the time of rating. Lets check if the average rating moves over time.

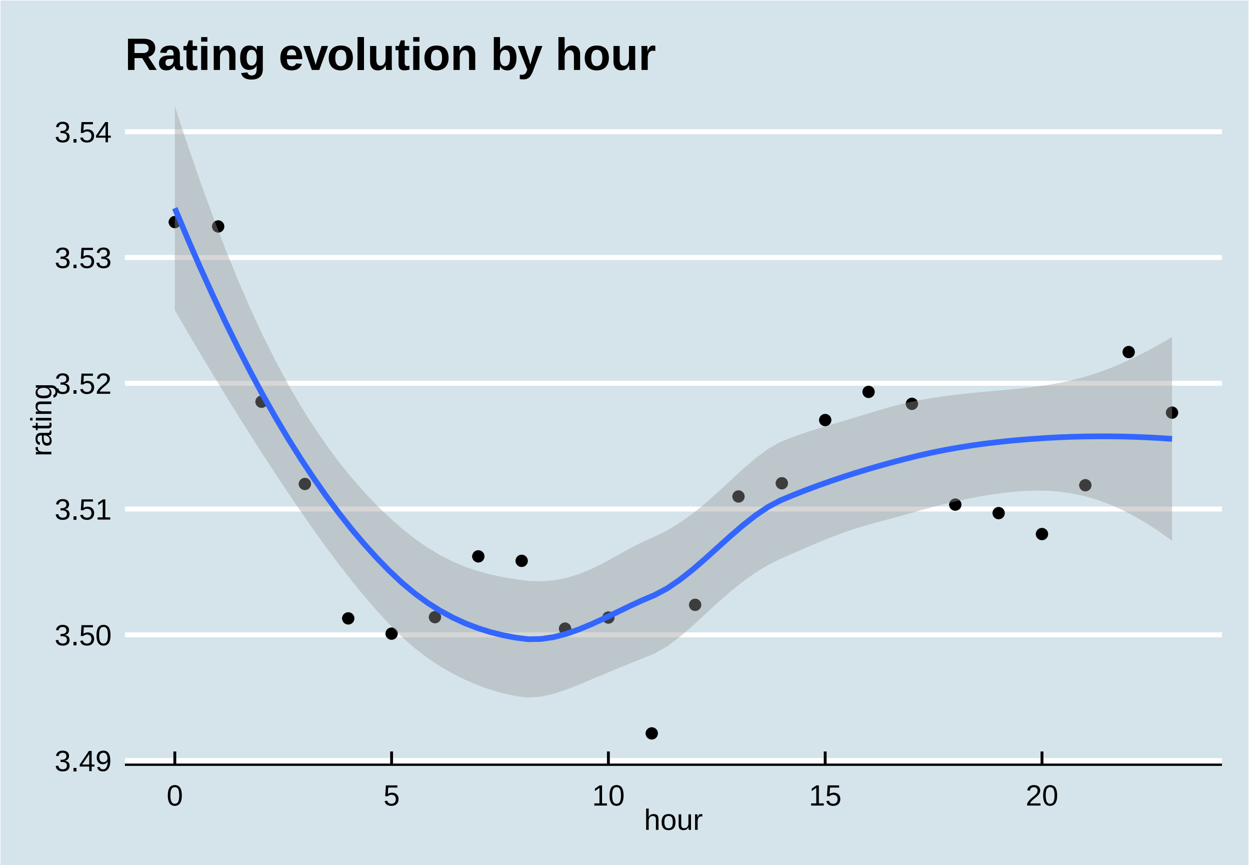
```
#Here, we use functions from the lubridate package to extract the week from the timestamp
edx %>%
  mutate(week = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(week) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(week, rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Rating evolution by week")+
  theme_economist()
```



Plotting the rating week wise, we observe a very weak effect, if any. The rating decreases slightly and then increases near the end of the timeline. We will not be incorporating the week effect into our model.

let us check if the hour of day has an effect on ratings.

```
edx %>%
  mutate(hour = hour(as_datetime(timestamp))) %>%
  group_by(hour) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(hour, rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Rating evolution by hour")+
  theme_economist()
```



The hour seems to have a significant inpact on the rating. Presumably, people who choose to watch movies at very late hours are harsher critics.

Data Preprocessing

In this section, we will address all of the wrangling and cleaning required to apply algorithms to our data.

In order to use the genre data in our model, it is necessary to decode it from the provided variable for the training, test and validation set. Similar to the exploration code, the following chunk puts a zero or one based on if the particular genre exists for an observation. We do this decoding before partitioning the data to avoid doing it on multiple dataframes.

```
#We choose 0 or 1 as the output since this will make it easier to make a linear model

data <-  edx %>% mutate(Romance = ifelse(str_detect(genres,"Romance"),1,0),
  Comedy = ifelse(str_detect(genres,"Comedy"),1,0),
  Action = ifelse(str_detect(genres,"Action"),1,0),
  Crime = ifelse(str_detect(genres,"Crime"),1,0),
  Thriller = ifelse(str_detect(genres,"Thriller"),1,0),
  Drama = ifelse(str_detect(genres,"Drama"),1,0),
  Sci_Fi = ifelse(str_detect(genres,"Sci-Fi"),1,0),
  Adventure = ifelse(str_detect(genres,"Adventure"),1,0),
  Children = ifelse(str_detect(genres,"Children"),1,0),
  Fantasy = ifelse(str_detect(genres,"Fantasy"),1,0),
  War = ifelse(str_detect(genres,"War"),1,0),
  Animation = ifelse(str_detect(genres,"Animation"),1,0),
  Musical = ifelse(str_detect(genres,"Musical"),1,0),
  Western = ifelse(str_detect(genres,"Western"),1,0),
  Mystery = ifelse(str_detect(genres,"Mystery"),1,0),
  Film_Noir = ifelse(str_detect(genres,"Film-Noir"),1,0),
  Horror = ifelse(str_detect(genres,"Horror"),1,0),
  Documentary = ifelse(str_detect(genres,"Documentary"),1,0),
  IMAX = ifelse(str_detect(genres,"IMAX"),1,0)
  )

glimpse (data)
```

```
## Rows: 9,000,055
## Columns: 25
## $ userId      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2~
## $ movieId     <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420~
## $ rating      <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5~
## $ timestamp   <int> 838985046, 838983525, 838983421, 838983392, 838983392, 838~
## $ title       <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", ~
## $ genres      <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|S~
## $ Romance     <dbl> 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0~
## $ Comedy      <dbl> 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0~
## $ Action      <dbl> 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1~
## $ Crime       <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Thriller    <dbl> 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Drama       <dbl> 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1~
## $ Sci_Fi      <dbl> 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0~
## $ Adventure   <dbl> 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0~
## $ Children    <dbl> 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0~
## $ Fantasy     <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0~
## $ War         <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1~
## $ Animation   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0~
## $ Musical     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0~
## $ Western     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Mystery     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Film_Noir   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Horror      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Documentary <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ IMAX        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

```
#If we choose this model, we will have to apply the same steps to the validation set.
```

The first step of implementing a machine learning algorithm is to create training and test sets. We do this using the caret package function createDataPartition(). We then use the semi_join function to ensure all items in the test set exist in the training set also.

```
#We divide the data into training and test sets.
#Before this, we set a seed to ensure that the work is reproducible
#The index already made in the first code chunk was to separate
#the validation set so we need to remake the index



#We choose an 80/20 split, the standard in ML
```

```
test_index <- createDataPartition(y = data$rating, times = 1, p = 0.2, list = FALSE)

train_set <- data[-test_index,]
test_set <- data[test_index,]

#We don't want movies or users in the test set that don't appear in the training set

test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

#Validation set was already created in the code chunk Initial dataframes
```

In order to use models from reccomenderlab, we need to convert our data to into a sparse matrix format where users are rows, movies are columns and NAs are stored much more efficiently than in a data.frame.This is accomplished using the sparseMatrix function from the Matrix package. I chose to make a function so that we can create the sparse matrix before trying any algorithms.

```
#For using recommenderlab models, we need to make a sparse matrix and partition it
#using the inbuilt partition functions

create_sparse_m <- function(){

  sparse_m <- sparseMatrix(
    i = as.numeric(as.factor(edx$userId)),
    j = as.numeric(as.factor(edx$movieId)),
    x = edx$rating,
    dims = c(length(unique(edx$userId)),
             length(unique(edx$movieId))),
    dimnames = list(paste("u", 1:length(unique(edx$userId)), sep = ""),
                    paste("m", 1:length(unique(edx$movieId)), sep = "")))

  sparse_m <- new("realRatingMatrix", data = sparse_m)
  return (sparse_m)

}
```

In training algorithms on recommenderlab, it was observed that it takes a massive amount of time for collaborative filtering algorithms to train. Some reduction in data was required. We had already observed from the first section that some part of data is much more pertinent than the other. This reduced scope of data will only apply to algorithms that take too much time to train. We will examine the effect of this data reduction on the RMSE while fine tuning the recommenderlab UBCF model. The following is an example of the code required to reduce the data. For reduction, we select a percentile of the most pertinent users and movies.

```
  #90 percentile data selection
  sparse_m <- create_sparse_m()
  y = 0.9
  min_movies_by_user <- quantile(rowCounts(sparse_m),y)
  min_ratings_per_movie<- quantile(rowCounts(sparse_m),y)
```

In order to evaluate different models, we will be using the RMSE (Root Mean Squared Error). There are other more sophisticated measures of a models performance like MAE (Mean Absolute Error) which gives less weight to outliers but we will not be using them in this analysis. In the following code, we write a function to calculate the RMSE.

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Methods and Analysis

We have established that there is a user effect, movie effect, genre effect that we want to calculate. we will follow the approach used by R Irizarry in his book at https://rafalab.github.io/dsbook/recommendation-systems.html and build on that by adding the genre effect. Afterwards, we will test collaborative filtering algorithms from the recomenderlab package and the recosystem package noting all otheir RMSEs when evaluated on our test data. In the end, we will select the best algorithms and apply them on our validation set to choose a winner.

During model building, we must use only the train_set.

Regression Models using movie, user, genre, time The most simplistic model envisagable consists of predicting the mean for all the ratings. This can be described like this:

$$Y_{u,i} = \mu + \varepsilon_{u,i} \tag{1}$$

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

#We make a a dataframe as long as test_set with mu repeated as R Markdown
#interprets the difference in length between mu and test_set as an error.

  naive_rmse <- RMSE(test_set$rating,rep(mu,nrow(test_set)))
  naive_rmse
```

```
## [1] 1.060704
```

```
  rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
```

To calculate the movie and user effects, we could use the lm function. However, because there are 1000 + movies and 60000 + users, the lm function is very expensive to use, computationally. Instead, we will estimate these effects in the following way. In a model with movie effect (b_i) + user effect (b_u),

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i} \tag{2}$$

Let us start with calculating b_i and b_u and testing our model.

Here b_i will be average of $Y_{u,i} - \mu$ and b_u will be average of $Y_{u,i} - \mu - b_i$

```
knitr::opts_chunk$set(cache = TRUE)

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

```
user_avgs <- train_set %>%
  left_join(movie_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u= mean(rating-b_i-mu))

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

User_And_Movie_effect <- RMSE(test_set$rating,predicted_ratings)

User_And_Movie_effect
```

```
## [1] 0.8661625
```

```
rmse_results <- add_row(rmse_results,method = "Movie Average and User Average",
                        RMSE = User_And_Movie_effect)
```

We observe a significant improvement, taking our RMSE into the 0.8s. However, because of the very large number of movies and users, this model is prone to errors occuring due to obscure movies that have few data points and thus, ideally should be considered less pertinent to our analysis. The following code illustrates our errors.

```
#Checking errors to see which are the one we got wrong
nmbr_ratings <- data %>% group_by(movieId) %>% summarise(n=n())
test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by="userId") %>%
  left_join(nmbr_ratings, by = "movieId") %>%
  mutate(residual = rating - (mu + b_i + b_u)) %>%
  arrange(desc(abs(residual))) %>%
  slice(1:10) %>%
  select(title, residual, n) %>%
  knitr::kable()
```

title	residual	n
Gigli (2003)	5.130790	313
Lord of the Rings: The Two Towers, The (2002)	-5.034597	12969
Million Dollar Duck, The (a.k.a. $1,000,000 Duck) (1971)	4.902788	151
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)	-4.880573	10627
Meatballs 4 (1992)	4.843295	114
Schindler's List (1993)	-4.840842	23193
Double Indemnity (1944)	-4.821004	2154
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	-4.820241	2922
Downfall (Der Untergang) (2004)	-4.796959	1212
Cannonball Run III (a.k.a. Speed Zone!) (1989)	4.757313	138

The majority of the errors are caused by obscure movies. We thus should penalize movies with very few ratings (obscure movies) and users with very few ratings (inexperienced users). We do this by regularization.

In this approach, we penalize obscure movies and inexperienced users by altering the mean equation in adding a penalty term to the denominator. Here is the equation.

$$\frac{1}{N}\sum_{u,i}\left(y_{i,u}-\mu-b_i-b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right) \tag{3}$$

Using calculus, it can be shown that the values of b that minimize this equation are:

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

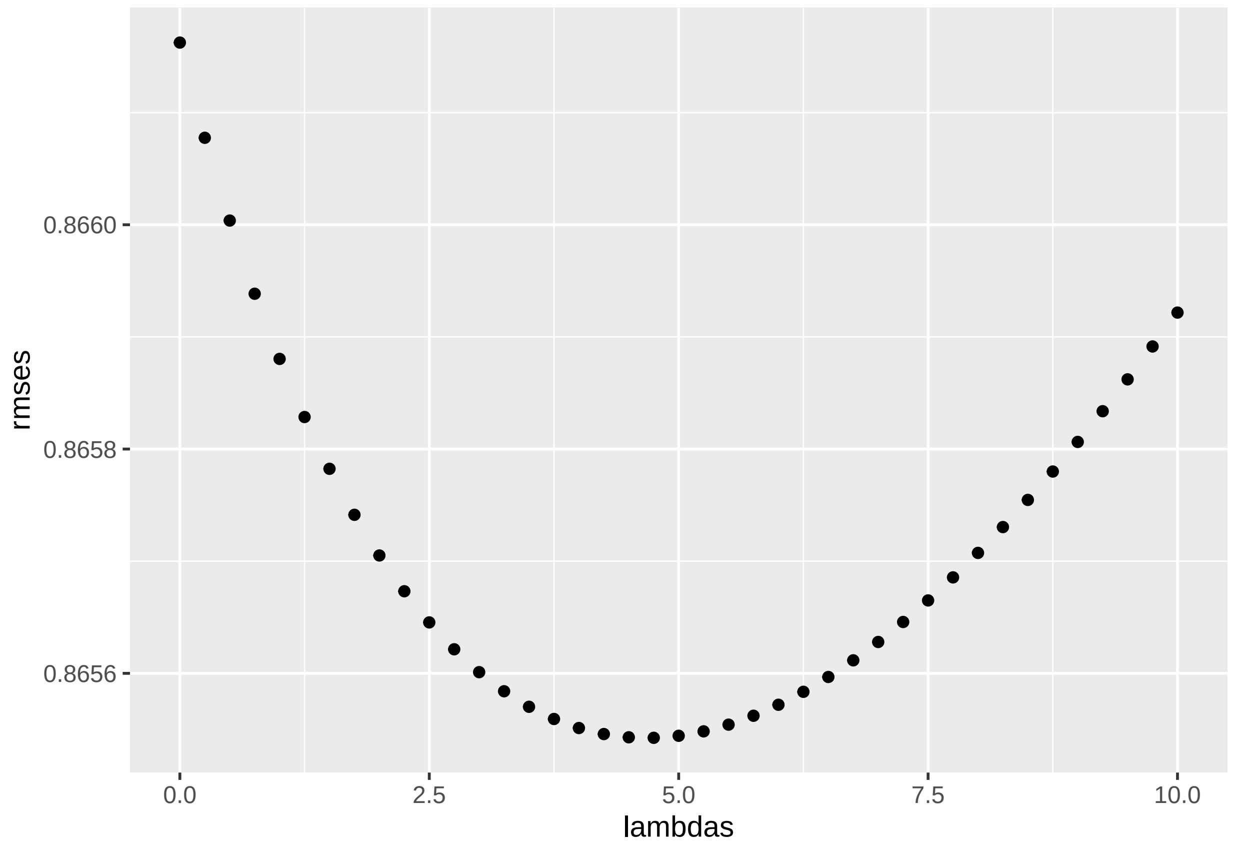
However, we need to choose a λ carefully as we don't want to penalize prolific users or good movies. λ thus becomes a tuning parameter and we can run a loop to identify the value which minimizes our RMSE.

The following code does that and plots λ against the rmse.

```
knitr::opts_chunk$set(cache = TRUE)

lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})

lambda <- lambdas[which.min(rmses)]
b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

gc()
```

```
##              used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells   3231316  172.6    5462340   291.8    5462340   291.8
## Vcells 631073893 4814.8 1377090314 10506.4 1372940331 10474.8
```

```
qplot(lambdas,rmses)
```



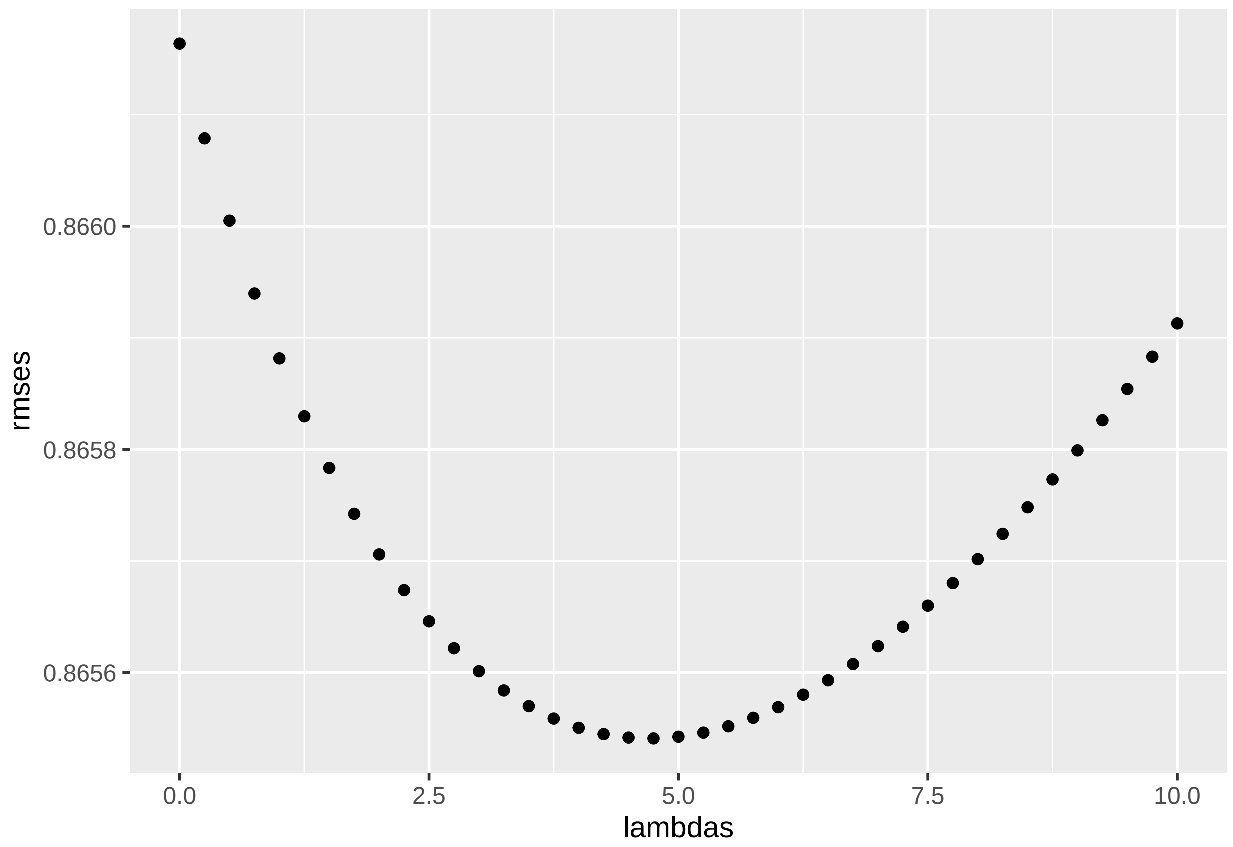
We can see from the above plot that we get our minumum RMSE for λ between 3 and 5. Lets add the minimum RMSE to our record.

```
rmse_results <- rmse_results %>% add_row(method ="Regularized user & Movie effects", RMSE =min(rmses))
rmse_results
```

```
## # A tibble: 3 x 2
##   method                            RMSE
##   <chr>                            <dbl>
## 1 Just the average                 1.06
## 2 Movie Average and User Average   0.866
## 3 Regularized user & Movie effects 0.866
```

We can add in the same way the time effect. Since fewer movies will be rated at night, we can subject time to the same regularization.

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

rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  b_t <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    mutate(hour = hour(as_datetime(timestamp))) %>%
    group_by(hour) %>%
    summarize(b_t = sum(rating - mu - b_i - b_u)/(n()+l) )

  predicted_ratings <-
    test_set %>%
    mutate(hour = hour(as_datetime(timestamp))) %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "hour") %>%
    mutate(pred = mu + b_i + b_u + b_t) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})


lambda <- lambdas[which.min(rmses)]

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

b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu -b_i)/(n()+lambda))

b_t <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  mutate(hour = hour(as_datetime(timestamp))) %>%
  group_by(hour) %>%
  summarize(b_t = sum(rating - mu - b_i - b_u)/(n()+lambda) )

qplot(lambdas,rmses)
```



```
gc()#Garbage collection
```

```
##             used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells   3261720  174.2    5462340   291.8    5462340   291.8
## Vcells 631155074 4815.4 1652588376 12608.3 1652587099 12608.3
```

```
rmse_results <- rmse_results %>% add_row(method ="Regularized user,movie,time effects",
                                          RMSE =min(rmses))
```

While the number of movies and users is very high, the number of genres is low at 19. We can thus use the lm() function to make a linear regression model for calculating the genre effect. This will have to be based on the residual value leftover from calculating the movie effect and user effect. Each observation has 0 or 1 for each genre (k) . Coefficients calculated by the linear model c_g for each genre will be multiplied and added for each row. Our final regression model, thus will look like this.

$$\frac{1}{N}\sum_{u,i}(y_{i,u}-\mu-b_i-b_u-b_t)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2+\sum_t b_t^2\right)+\sum_g k_g c_g \qquad (3)$$

```
invisible(gc())
genre_train <- train_set %>%
  mutate(hour = hour(as_datetime(timestamp))) %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_t, by = "hour") %>%
  mutate(residual = rating - mu- b_i-b_u-b_t )

temp <- genre_train %>% select(c(Romance:IMAX,residual))
lm_fit <-   lm(residual ~ ., data=temp)

temp2 <- test_set %>% select(c(Romance:IMAX))

pred <- predict.lm(lm_fit, newdata = temp2)

predicted_ratings <-
  test_set %>%
  mutate(hour = hour(as_datetime(timestamp))) %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId")%>%
  left_join(b_t, by = "hour") %>%
  mutate(pred = mu + b_i + b_u + b_t + pred) %>%
  pull(pred)


rmse_genre_effect <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- rmse_results %>% add_row(method ="Regularized user,movie,time effects with Genre (lm)",
                                         RMSE =min(rmse_genre_effect))

rm(temp,temp2)

rmse_results
```

```
## # A tibble: 5 x 2
##   method                                                RMSE
##   <chr>                                                <dbl>
## 1 Just the average                                     1.06
## 2 Movie Average and User Average                       0.866
## 3 Regularized user & Movie effects                     0.866
## 4 Regularized user,movie,time effects                  0.866
## 5 Regularized user,movie,time effects with Genre (lm) 0.865
```

We can also check the linear modeling coefficients produced by our mode like this.

```
summary(lm_fit)$coefficients
```

```
##                 Estimate   Std. Error     t value     Pr(>|t|)
## (Intercept)  0.000173412 0.0009686883   0.1790174 8.579241e-01
## Romance     -0.007853563 0.0008632207  -9.0979783 9.205129e-20
## Comedy      -0.001118146 0.0008046927  -1.3895313 1.646713e-01
## Action      -0.010257563 0.0009049362 -11.3351224 8.796027e-30
## Crime        0.006957691 0.0009938950   7.0004283 2.552034e-12
## Thriller    -0.008330430 0.0008826340  -9.4381484 3.795315e-21
## Drama        0.015288065 0.0007958640  19.2093935 3.103249e-82
## Sci_Fi      -0.004433596 0.0010015151  -4.4268884 9.560358e-06
## Adventure   -0.004033054 0.0009337896  -4.3190184 1.567268e-05
## Children    -0.031706385 0.0016546032 -19.1625311 7.644080e-82
## Fantasy      0.006584809 0.0011481752   5.7350210 9.750413e-09
## War         -0.001865583 0.0014410102  -1.2946357 1.954460e-01
## Animation    0.013266341 0.0019373412   6.8477047 7.505025e-12
## Musical     -0.001599133 0.0016457190  -0.9716927 3.312035e-01
## Western     -0.004666388 0.0022532599  -2.0709498 3.836352e-02
## Mystery      0.010914443 0.0013990088   7.8015544 6.115734e-15
## Film_Noir    0.024718021 0.0028885991   8.5570965 1.157677e-17
## Horror       0.007279741 0.0012954916   5.6192881 1.917529e-08
## Documentary  0.061622721 0.0032623849  18.8888564 1.414918e-79
## IMAX        -0.020028384 0.0106219557  -1.8855647 5.935368e-02
```

We see that we have managed to reduce the RMSE to 0.8650. We will now explore packaged collaborative filtering algorithms from recommenderlab and recosystem

Recommendation Engines According to https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf ,collaborative filtering (CF) uses given rating data by many users for many items as the basis for predicting missing ratings or creating topN lists. Collaborative filtering algorithms are typically divided into two groups, memory-based CF and model-based CF algorithms (Breese, Heckerman, and Kadie 1998). Memory-based CF use the a large sample of the user database to create recommendations.

Recommenderlab models The most prominent CF algorithm is the user-based collaborative filtering which operates under the assumption that users with similar preferences will rate items similarly. These algorithms calculate either the Pearson or Cosine similarity between users and then aggregate their ratings (simplest would be to average) to guess the ratings that users have not based based on their similarity to others who have made the ratings.

A prominent problem of this approach is scalability : the whole database needs to be processed for creating recommendations. For our analysis, we will be using a 90 percentile part of the dataset.

The UBCF algorithm is very slow to compute. It has 3 tuning parameters. nn is the number of nearest neighbours which make up a neigbourhoood. k is the number of given elements from which a similar rating will be claculated. The following code shows how to train the UBCF algortihm. Note that we are not using our test and train variables as recommenderlab doesn't allow for them to be used externally. Tuning code has not been included as it takes too much time to run.

```r
knitr::opts_chunk$set(cache = TRUE)

  y=0.9
  sparse_m <- create_sparse_m()
  nn=15
  set.seed(1991)
  min_movies_by_user <- quantile(rowCounts(sparse_m),y)
  min_ratings_per_movie<- quantile(rowCounts(sparse_m),y)
  sparse_m_limited <- sparse_m[rowCounts(sparse_m)>=min_movies_by_user,
                               colCounts(sparse_m)>=min_ratings_per_movie]
  e4 <- evaluationScheme(sparse_m_limited , method="split", train=0.9,
                         k=1, given=8)
  r_UBCF <- Recommender(getData(e4, "train"), "UBCF",parameter=c(nn=nn))
  p_UBCF <- predict(r_UBCF, getData(e4, "known"), type="ratings")
  rmse_UBCF <- calcPredictionAccuracy(p_UBCF, getData(e4, "unknown"))[1]

  rmse_results <- rmse_results %>% add_row(method ="UBCF using reccomenderlab",
                                           RMSE =rmse_UBCF)
```

A sister to the UBCF is the IBCF algorithm which creates a similarity matrix between items. The reasoning is that if a user likes certain items, he will like other items similar to it. It is similary slow to compute. Here is the code to implement it

```
knitr::opts_chunk$set(cache = TRUE)
gc()#Garbage collection
```

```
##              used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells    3286000  175.5    5462340   291.8    5462340   291.8
## Vcells 1174264419 8959.0 1983186051 15130.6 1652587099 12608.3
```

```
  y=0.9
  sparse_m <- create_sparse_m()
  nn=15
  set.seed(1991)
  min_movies_by_user <- quantile(rowCounts(sparse_m),y)
  min_ratings_per_movie<- quantile(rowCounts(sparse_m),y)
  sparse_m_limited <- sparse_m[rowCounts(sparse_m)>=min_movies_by_user,
                               colCounts(sparse_m)>=min_ratings_per_movie]
  model_ibcf <- Recommender(sparse_m_limited, method="IBCF", param=list(normalize="center"))
  #Testing prediction
  #pred_pop <- predict(model_ibcf, sparse_m[1:10], type="ratings")

  #as(pred_pop, "matrix")[,1:10]
  #Finding RMSE
  e6 <- evaluationScheme(sparse_m_limited , method="split", train=0.9,
                         k=1, given=8)
  p_IBCF <- predict(model_ibcf, getData(e6, "known"), type="ratings")
  rmse_IBCF <- calcPredictionAccuracy(p_IBCF, getData(e6, "unknown"))[1]

  rmse_results <- rmse_results %>% add_row(method ="IBCF using reccomenderlab", RMSE =rmse_IBCF)
```

Another algorithm in reccomenderlab is the POPULAR algorithm which is based on item popularity

```
knitr::opts_chunk$set(cache = TRUE)
  #Let us try the POPULAR method
gc()#Garbage collection
```

```
##              used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells    3291905  175.9    5462340   291.8    5462340   291.8
## Vcells 1190940545 9086.2 1983186051 15130.6 1983138286 15130.2
```

```
sparse_m <- create_sparse_m()
#Reduction parameter
y=0.9
min_movies_by_user <- quantile(rowCounts(sparse_m),y)
min_ratings_per_movie<- quantile(rowCounts(sparse_m),y)

sparse_m_limited <- sparse_m[rowCounts(sparse_m)>=min_movies_by_user,
                             colCounts(sparse_m)>=min_ratings_per_movie]
#Making the model
model_popular <- Recommender(sparse_m_limited, method="POPULAR", param=list(normalize="center"))
```

```
#Testing prediction

#Finding RMSE
e5 <- evaluationScheme(sparse_m_limited , method="split", train=0.9,
                       k=1, given=8)
p_POPULAR <- predict(model_popular, getData(e5, "known"), type="ratings")
rmse_POP <- calcPredictionAccuracy(p_POPULAR, getData(e5, "unknown"))[1]


rmse_results <- rmse_results %>% add_row(method ="POPULAR using reccomenderlab", RMSE =rmse_POP)
```

Recommenderlab contains many other algorithms such as FSVD Approximation and RANDOM items but we are not going to train them here. we have already seen that UBCF and IBCF performed poorly on our dataset with POPULAR being the only algorithm below 1 for RMSE.

Recosystem model Another recommendation algorithm is available in the recosystem package. According to https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html, 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 (https://www.csie.ntu.edu.tw/∼cjlin/libmf/).LIBMF is a high-performance C++ library for large scale matrix factorization. Matrix factorization tries to approximate the whole rating matrix $R_{mÖn}$ as the product of two matrices of lower dimensions, $P_{kÖm}$ and $QkÖn$, such that $R \approx P^{-1}Q$. The process of solving the matrices P and Q constitutes the model training. The solving method used in this package is called the Stochiastic Gradient Method to Matrix factorization.

```
knitr::opts_chunk$set(cache = TRUE)

gc()#Garbage collection
```

```
##               used    (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells   10168491   543.1   24908431  1330.3   10324796   551.5
## Vcells 1334735333 10183.3 1899711622 14493.7 1400972089 10688.6
```

```
#we select only the userID, MovieID and rating in the next three statements
test_GD <- as.matrix (test_set [,1:3])
train_GD <- as.matrix(test_set [,1:3])


set.seed(1)

#We need to build data stream objects as these are accepted inputs for this algorithm
train_GD_2 <- data_memory(train_GD[,1],train_GD[,2],train_GD[,3])
test_GD_2 <- data_memory(test_GD[,1],test_GD[,2],test_GD[,3])


#Next step is to build Recommender object
r = Reco()
# Matrix Factorization :  tuning training set
# lrate is the gradient descend step rate
# dim are the number of latent factors
# nthread is number of threads to use :
#REDUCE IF YOUR PROCESSOR DOESN'T SUPPORT 6 THREADS
```

```
opts = r$tune(train_GD_2, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                      costp_l1 = 0, costq_l1 = 0,
                                      nthread = 6, niter = 10))


r$train(train_GD_2, opts = c(opts$min, nthread = 6, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       1.2356   4.1293e+06
##    1       0.9122   2.8227e+06
##    2       0.8891   2.7232e+06
##    3       0.8780   2.6808e+06
##    4       0.8666   2.6508e+06
##    5       0.8527   2.6188e+06
##    6       0.8397   2.5870e+06
##    7       0.8293   2.5672e+06
##    8       0.8202   2.5482e+06
##    9       0.8113   2.5346e+06
##   10       0.8022   2.5184e+06
##   11       0.7934   2.5051e+06
##   12       0.7843   2.4924e+06
##   13       0.7755   2.4811e+06
##   14       0.7669   2.4697e+06
##   15       0.7584   2.4587e+06
##   16       0.7507   2.4503e+06
##   17       0.7435   2.4412e+06
##   18       0.7367   2.4340e+06
##   19       0.7301   2.4263e+06
```

```
pred <- r$predict(test_GD_2, out_memory())

rmse_MFGD <- RMSE(pred,test_set$rating)

rmse_results <- rmse_results %>% add_row(method ="Matrix Factorization using recosystem",
                                        RMSE =rmse_MFGD)

rmse_results
```

```
## # A tibble: 9 x 2
##   method                                                   RMSE
##   <chr>                                                   <dbl>
## 1 Just the average                                        1.06
## 2 Movie Average and User Average                          0.866
## 3 Regularized user & Movie effects                        0.866
## 4 Regularized user,movie,time effects                     0.866
## 5 Regularized user,movie,time effects with Genre (lm) 0.865
## 6 UBCF using reccomenderlab                               1.06
## 7 IBCF using reccomenderlab                               1.75
## 8 POPULAR using reccomenderlab                            0.904
## 9 Matrix Factorization using recosystem                   0.711
```

Results

- Matrix factorization using recosystem clearly beat all the algorithms we have tested so far giving us an RMSE of under 0.78 on our test data.
- We were also able to achieve very good results with our linear regression approach.
- Successive linear regression didn't yeild lot of quantitative returns with respect to the RMSE.
- Collaborative filtering models were computationally very expensive but didn't perform as well as linear regression

Final Evaluation and conclusion

RMSE on holdout set We will now calculate the RMSE on the holdout validation set using the best algorithm to arrive at our final RMSE. The algorithm "Matrix Factorization using recosystem" is chosen as the best performing with an RMSE of 0.710 on the test_set. This will be used for calculating the validation RMSE.

```
knitr::opts_chunk$set(cache = TRUE)

gc()#Garbage collection
```

```
##              used    (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells   10225780   546.2   24908431  1330.3   10552139   563.6
## Vcells 1351964306 10314.7 2279733946 17393.0 1400972089 10688.6
```

```
invisible(gc())
#Selecting only pertinent columns
valid_GD <-as.matrix(validation [,1:3])
valid_GD_2 <- data_memory(valid_GD[,1],valid_GD[,2],valid_GD[,3])

#selecting to output to variable
pred <- r$predict(valid_GD_2, out_memory())

rmse_MFGD_final <- RMSE(pred,validation$rating)

rmse_results <- rmse_results %>% add_row(
  method ="Matrix Factorization using recosystem : FINAL Validation RMSE ",
                                         RMSE =rmse_MFGD_final)

options(pillar.sigfig = 10)
rmse_results
```

```
## # A tibble: 10 x 2
##    method                                                                   RMSE
##    <chr>                                                                   <dbl>
##  1 "Just the average"                                               1.060704445
##  2 "Movie Average and User Average"                                 0.8661624502
##  3 "Regularized user & Movie effects"                               0.8655425098
##  4 "Regularized user,movie,time effects"                            0.8655410689
##  5 "Regularized user,movie,time effects with Genre (lm)"            0.8653895229
##  6 "UBCF using reccomenderlab"                                      1.056131289
##  7 "IBCF using reccomenderlab"                                      1.750012583
##  8 "POPULAR using reccomenderlab"                                   0.9035966682
##  9 "Matrix Factorization using recosystem"                          0.7109734786
## 10 "Matrix Factorization using recosystem : FINAL Validation RMSE " 0.8586301599
```

Conclusion The Grouplens dataset lends itself readily to exploring recommendation algorithms. In this report, we have discussed two linear regression approaches as well as tested the applicability of collaborative filtering and matrix factorization algorithms. We have found that recosystems Matrix Factorization using Stochastic Gradient is a potent algorithm for solving recommendation problems in an efficient manner.

Although we used cross validation to tune the parameters of the UBCF algorithm, we selected only defaults for others. Further reports could focus on better tuning collaborative filtering algorithms to minimize the RMSE. We could extract the year from the title to see if older or newer movies have any pattern in their ratings. For our regression model, We could select lambda individually for each effect. We could also change the order of our linear regression effects to see its effect on the RMSE. Other future work could also include ensemble methods using multiple algorithms and weighting them into a combined result.

The RMSE of Matrix Factorization using recosystem on the validation set is 0.85796, beating the target of 0.86490. The accompanying script file contains also the code to apply the final regression model to the validation set.

References

- https://rafalab.github.io/dsbook/
- https://www.inf.unibz.it/~ricci/papers/introduction-handbook-2015.pdf
- https://www.asc.ohio-state.edu/statistics/dmsl/GrandPrize2009_BPC_BellKor.pdf
- https://cran.r-project.org/web/packages/recommenderlab/recommenderlab.pdf
- https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf
- https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html