Project 1 : Movie Recommendation Engine

[redacted]

11/9/2020

Getting Started

Importing the Data

For this recommender the first thing we need to do is make sure that we have all the data we need. There are two datasets in this project: movies and ratings. Movies contains a movie id, the title of the movie, and the genre in `|` delineated lists. Ratings contains the user id making the rating, the movie id they are rating, and the timestamp of the rating. This dataset is a version of movieLens converted to a dataFrame. MovieLens is set as the type of realRatingMatrix. We can convert between the two types as I will show later in the project.

```r
library(readr)
library(recommenderlab)
data.movies <- read_csv("movies.csv")
data.ratings <- read_csv("ratings.csv")
data("MovieLense")
```

Importing the Libraries

We're going to need a few different libraries for this project that are all very crucial to the success of this project. The first and foremost library that we need is `recommenderlab`, this gives us the foundation we will need to build our recommendation system, and we use it to get the dataset `MovieLens`. The next is `ggplot2`, this will give us powerful data visualization tools. To transform the data between the `dataFrame` and `realRatingMatrix`, we will need both `data.table` and `reshape2`. To make transformations to the genre column we will need to import the `dplyr` and `tidyr` libraries. Each of these libraries vastly simplify the operations we are completing in this project, and help the code be more succint and maintainable.

```r
library(ggplot2)
library(data.table)
library(reshape2)
library(dplyr)
library(tidyr)
```

Quick Overview of the Data

Before we jump into processing the data, let's familiarize ourselves with the data we just imported.

Summary of Movie Data

There are three fields in movies: `movieId`, `title`, and `genres`. `movieId` is a numerical value denoted the identifying value of the movie record. `title` is eponymously the title of the movie in the record. `genres` is going to be tricky, it's a string of all the movie delineated by `|`. This column is not formatted for optimal processing because the computer can not parse the formating of the genre in a useful manner yet, we will have to fix this in the data pre-processing.

```
summary(data.movies)
```

```
##     movieId           title              genres
##  Min.   :     1   Length:10329       Length:10329
##  1st Qu.:  3240   Class :character   Class :character
##  Median :  7088   Mode  :character   Mode  :character
##  Mean   : 31924
##  3rd Qu.: 59900
##  Max.   :149532
```

```
head(data.movies, n = 2)
```

```
## # A tibble: 2 x 3
##   movieId title            genres
##     <dbl> <chr>            <chr>
## 1       1 Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
## 2       2 Jumanji (1995)   Adventure|Children|Fantasy
```

Summary of the Ratings Data

We have four columns of data in this set: `userId`, `movieId`, `rating`, and `timestamp`. The userId is a unique identifier of the user. The `movieId` is an identifier **linked to the previous dataset** with the identifier value of the movie they are reviewing. `rating` is the rating the user gave, the scale is out of 5 with 5 beings the best rating the user can give. The timestamp is a UTC timestamp using the UNIX time stamping format. This format counting every millisec that has passed since 01/01/1970.

```
summary(data.ratings)
```

```
##      userId          movieId           rating        timestamp
##  Min.   :  1.0   Min.   :     1   Min.   :0.500   Min.   :8.286e+08
##  1st Qu.:192.0   1st Qu.:  1073   1st Qu.:3.000   1st Qu.:9.711e+08
##  Median :383.0   Median :  2497   Median :3.500   Median :1.115e+09
##  Mean   :364.9   Mean   : 13381   Mean   :3.517   Mean   :1.130e+09
##  3rd Qu.:557.0   3rd Qu.:  5991   3rd Qu.:4.000   3rd Qu.:1.275e+09
##  Max.   :668.0   Max.   :149532   Max.   :5.000   Max.   :1.452e+09
```

```
head(data.ratings, n = 2)
```

```
## # A tibble: 2 x 4
##   userId movieId rating  timestamp
##    <dbl>   <dbl>  <dbl>      <dbl>
## 1      1      16    4   1217897793
## 2      1      24    1.5 1217895807
```

Pre-Processing the Data

Genre

We identified in our data exploration process that the genre column was somewhat unusable for any computation or statistical process, which is what we are trying to accomplish here. So we need to process the genres into something usable. In one of our labs we had to do this, but the data there was categorical. This data is categorical with one major caveat **the data has multiple categories in a single column**. This is a big problem. The only way we can store this data in a meaningful way is to use "one-hot" variables to store each category as a column of binary values with one signifying membership in that genre.

The first step to accomplish this is to separate the genres into rows by the delimiting `|`. Then we're gonna put a one at the end of all data. Then we are going to spread the genres and values, so there are 1s and NAs. We then replace all NAs that are in numeric columns with 0s. We then clean some extraneous columns.

```
movies.genre <- data.movies %>% separate_rows(genres, sep = "[|]") %>% mutate(Value = 1) %>%
    spread(genres, Value) %>% mutate_if(is.numeric, ~replace(., is.na(.), 0))
movies.genre <- movies.genre[-c(3, 15)]
movies.genre
```

```
## # A tibble: 10,329 x 20
##    movieId title Action Adventure Animation Children Comedy Crime Documentary
##      <dbl> <chr>  <dbl>     <dbl>     <dbl>    <dbl>  <dbl> <dbl>       <dbl>
##  1       1 Toy ~      0         1         1        1      1     0           0
##  2       2 Juma~      0         1         0        1      0     0           0
##  3       3 Grum~      0         0         0        0      1     0           0
##  4       4 Wait~      0         0         0        0      1     0           0
##  5       5 Fath~      0         0         0        0      1     0           0
##  6       6 Heat~      1         0         0        0      0     1           0
##  7       7 Sabr~      0         0         0        0      1     0           0
##  8       8 Tom ~      0         1         0        1      0     0           0
##  9       9 Sudd~      1         0         0        0      0     0           0
## 10      10 Gold~      1         1         0        0      0     0           0
## # ... with 10,319 more rows, and 11 more variables: Drama <dbl>, Fantasy <dbl>,
## #   `Film-Noir` <dbl>, Horror <dbl>, Musical <dbl>, Mystery <dbl>,
## #   Romance <dbl>, `Sci-Fi` <dbl>, Thriller <dbl>, War <dbl>, Western <dbl>
```

Real Rating Matrix

As stated in the intoduction, the type of `realRatingMatrix` is important to the project, so much so that the original dataset is in this format. This format is not great for data exploration so I imported a version of the data that is in the `dataFrame` format and one that is in the `realRatingMatrix` type. This is a small guide as to how we would convert between the `dataFrame` and the `realRatingMatrix`. The `realRatingMatrix` is a sparse matrix that contains ratings as the primary value of focus. It's important we strip out the userId column otherwise we produce a myriad of errors that actually took an hour to solve.

```
matrix.rating <- dcast(data.ratings, userId ~ movieId, value.var = "rating")
matrix.rating <- as.matrix(matrix.rating[, -1])
matrix.rating <- as(matrix.rating, "realRatingMatrix")
matrix.rating
```

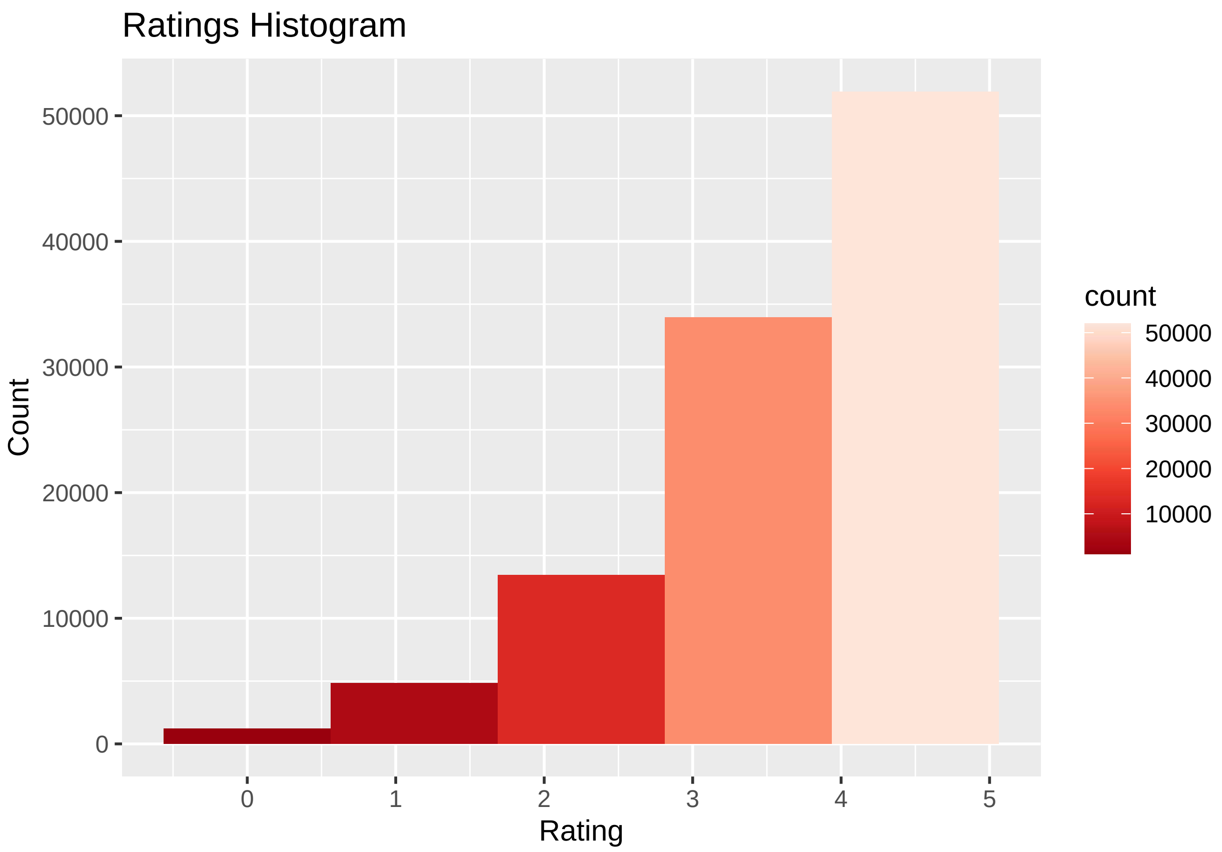
```
## 668 x 10325 rating matrix of class 'realRatingMatrix' with 105339 ratings.
```

Data Exploration

Ratings Overview

Let's dive deeper into the data. The first thing I want to look at is the overall distribution of the ratings in this dataset. Because I am only looking at one-dimensional data, I will use a histogram to accomplish this task. The ratings are out of five, so naturally we will use 5 bins in the histogram. This will give us a brief overview of the skew of the ratings.

```
ggplot(data = data.ratings) + geom_histogram(aes(x = rating, fill = ..count..), bins = 5) +
    scale_fill_distiller(palette = "Reds") + labs(x = "Rating", y = "Count", title = "Ratings Histogram
```



Looking at the data, the ratings skew towards the 5, meaning people are more likely to rate a movie well vs. critically.

Data Exploration

Top 10 Most Reviewed Movies

I wanted to take a peek at what movies had the most reviews. I wanted to do this to accomplish two goals: find the highest count of reviews on the top-`n` movies, and to see what movies were popular amoung the `MovieLens` reviewers.

```
movie.views <- data.ratings %>% group_by(movieId) %>% tally()
movie.views <- merge(movie.views, data.movies[, c("movieId", "title")])
movie.views <- movie.views[order(movie.views$n, decreasing = TRUE), ]
movie.views <- movie.views %>% select(-movieId)
head(movie.views, n = 10)
```

```
##        n                                      title
## 261  325                        Pulp Fiction (1994)
## 317  311                        Forrest Gump (1994)
## 280  308           Shawshank Redemption, The (1994)
## 427  294                       Jurassic Park (1993)
## 526  290           Silence of the Lambs, The (1991)
## 231  273 Star Wars: Episode IV - A New Hope (1977)
## 2057 261                         Matrix, The (1999)
## 523  253          Terminator 2: Judgment Day (1991)
## 99   248                          Braveheart (1995)
## 472  248                    Schindler's List (1993)
```

Number of Movies in Each Genre

```
genre_list <- colnames(movies.genre)[-(1:2)]

for (genre in genre_list) {

}
```

Model Creation

I haven't been able to properly introduce you to recommenderlab yet! Let's change that. So recommenderlab is a package that allows developers to quickly and easily develop and test recommender systems in R. You're probably thinking, "that's great but what does that actually look like?" With recommenderlab we get **11 different recommendation algorithms**, **3 different means of evaluation**, and **several different evaluation measures**.

Algorithm Selection

Of those 11 different algorithms, I selected `Item-Based Collaborative Filtering` as the most viable option for our model. This algorithm takes movies that are rated well by users of similar taste, and groups those users together recommending movies to other users in the group that have similar taste. Sounds familiar right? That's because this algorithm is a watered-down version of the real algorithm they use on many different streaming services! I think that's pretty cool, I hope you do too.

```
models.rec <- recommenderRegistry$get_entries(dataType = "realRatingMatrix")
models.rec$UBCF_realRatingMatrix$parameters
```

```
## $method
## [1] "cosine"
##
## $nn
## [1] 25
##
## $sample
## [1] FALSE
##
## $weighted
## [1] TRUE
##
## $normalize
## [1] "center"
##
## $min_matching_items
## [1] 0
##
## $min_predictive_items
## [1] 0
```

Training and Testing Data

In order to train the model and then make sure the model is fitted correctly to the data, we need to seperate the data we have in to training data, that will be used to fit the model, and testing data, that will be used to test the fit of the data within the model. We will be using an 80/20 split on training and testing data. We are using the realRatingMatrix from the original dataset to avoid overcomplicating the realRatingSection with the merging of the two dataFrames into a matrix.

```
data.movie.lens <- MovieLense
isTrain <- sample(x = c(TRUE, FALSE), size = nrow(data.movie.lens), replace = TRUE,
    prob = c(0.8, 0.2))
model.data.train <- data.movie.lens[isTrain, ]
model.data.test <- data.movie.lens[!isTrain, ]
```

Model Creation Cont.

Building the recommendation model

```
model.rec <- Recommender(data = model.data.train, method = "IBCF")
model.rec
```

```
## Recommender of type 'IBCF' for 'realRatingMatrix'
## learned using 747 users.
```

Model Results

Applying the Model to Some Testing Data

Now that we have a working model, let's use the testing data to try to predict what movies those users would like to see. We will pick the top 5 movies for them and look at the first 4 user predictions.

```
model.predict <- predict(object = model.rec, newdata = model.data.test, n = 5)
matrix.predict <- sapply(model.predict@items, function(x) {
    colnames(data.movie.lens)[x]
})
matrix.predict[, 1:4]
```

```
##      18
## [1,] "Desperado (1995)"
## [2,] "Haunted World of Edward D. Wood Jr., The (1995)"
## [3,] "Aliens (1986)"
## [4,] "Shall We Dance? (1996)"
## [5,] "Restoration (1995)"
##      21
## [1,] "Blue Angel, The (Blaue Engel, Der) (1930)"
## [2,] "Paths of Glory (1957)"
## [3,] "Fluke (1995)"
## [4,] "Afterglow (1997)"
## [5,] "Pest, The (1997)"
##      27
## [1,] "Deceiver (1997)"
## [2,] "Losing Chase (1996)"
## [3,] "Wedding Gift, The (1994)"
## [4,] "Maya Lin: A Strong Clear Vision (1994)"
## [5,] "Powder (1995)"
##      31
## [1,] "Antonia's Line (1995)"
## [2,] "Haunted World of Edward D. Wood Jr., The (1995)"
## [3,] "Maya Lin: A Strong Clear Vision (1994)"
## [4,] "Kansas City (1996)"
## [5,] "Spice World (1997)"
```

We can see that the model has produced the top five movies for those users, which atleast means our model is up and running. Quick note for anyone running the Rmd file, this knits but will oddly not work in the R console. Don't ask me why, I don't know.

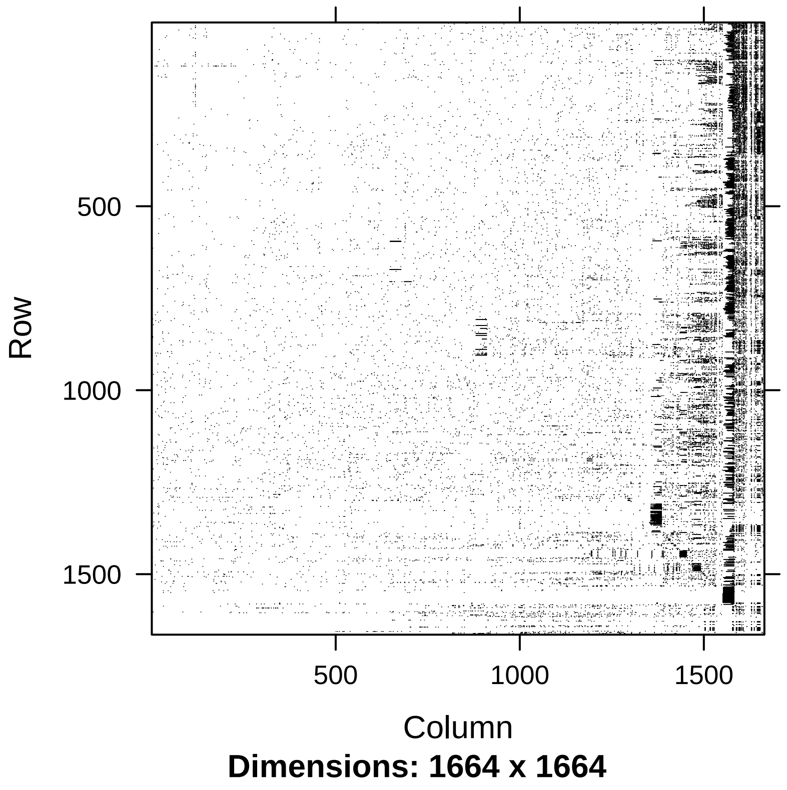
Model Results Cont.

Similarity Matrix

The way IBCF works is as follows: 1. Create a record that user u1 rated the following items: i1, i2, . . . , in. 2. Create a record for all users in the dataset like this 3. Find the similarities between the users

We will show this raw similarity matrix. It actually has already been generated as a part of the IBCF recommendation system.

```
model.info <- getModel(model.rec)
image(model.info$sim)
```



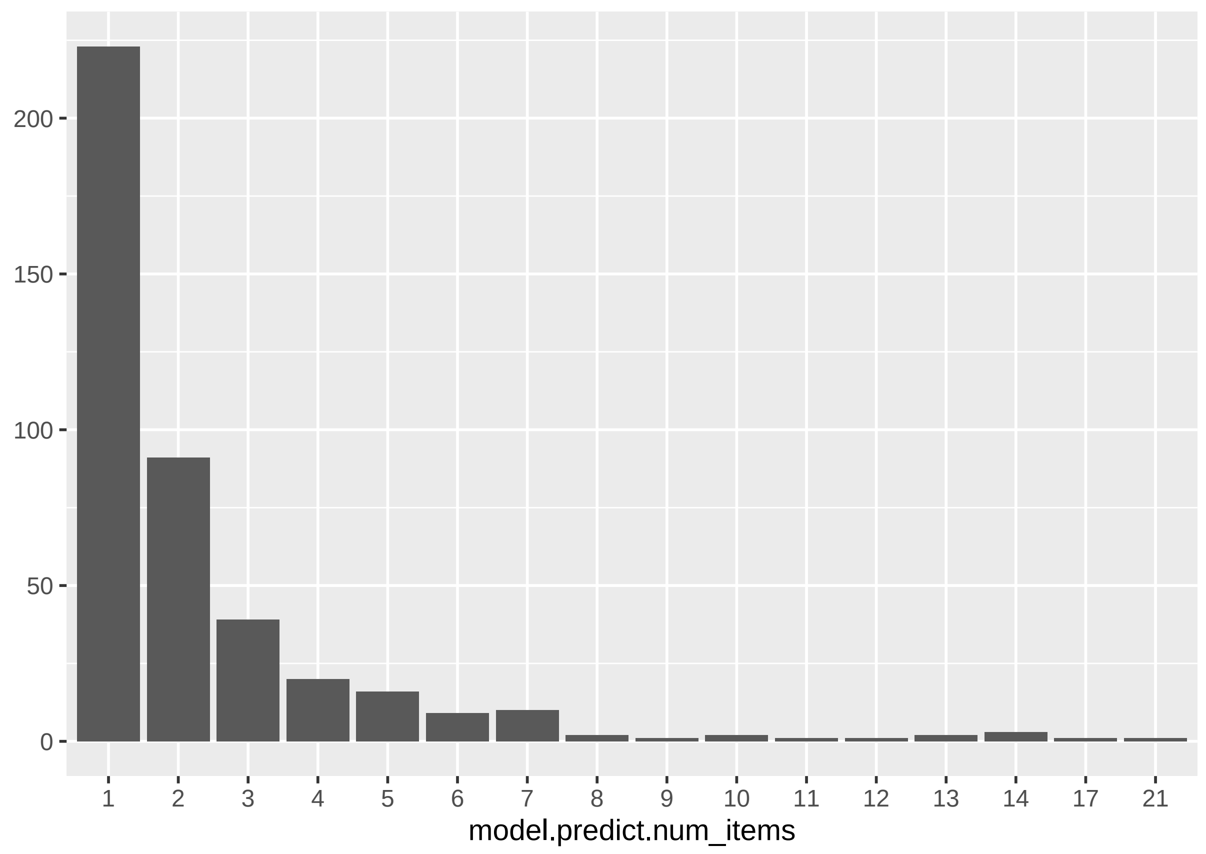
This may be faint but the darker the pixel to higher the similarity value between the two indexes (users). Some things we can glean from this data is the later movies in the matrix are somehow similar to nearly every movie in the dataset which may be indicative of faulty ratings or model overfit.

Model Results Cont.

Number of Times a Movie is Recommended

Some movies are more popular than others. This means the algorithm will recommend certain movies more than others. We can see that our data is long-tailed in that certain movies are wildly popular but a vast majority of movies are only recommended to under 5 users. This means that the algorithm is finding more unique movies to each different users taste, which is indicitive of a good model fit.

```r
model.predict.num_items <- factor(table(matrix.predict))
qplot(model.predict.num_items)
```



Model Results Cont.

Top 10 Recommended Movies

In the last bit of analysis of the model, I'd like to take a peek at the top 10 most recommended movies by the algorithm. We can do this easily by using the built in `sort` and `head` functions.

```r
model.predict.num_items.sorted <- sort(model.predict.num_items, decreasing = TRUE)
head(model.predict.num_items.sorted, n = 10)
```

```
## Horseman on the Roof, The (Hussard sur le toit, Le) (1995)
##                                                          21
##        When the Cats Away (Chacun cherche son chat) (1996)
##                                                          17
##   Flower of My Secret, The (Flor de mi secreto, La) (1995)
##                                                          14
##            Haunted World of Edward D. Wood Jr., The (1995)
##                                                          14
##                     Maya Lin: A Strong Clear Vision (1994)
##                                                          14
##                                              Nadja (1994)
##                                                          13
##                                                    unknown
##                                                          13
##                               Operation Dumbo Drop (1995)
##                                                          12
##                                        Blood Beach (1981)
##                                                          11
##                 Blue Angel, The (Blaue Engel, Der) (1930)
##                                                          10
## Levels: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 17 21
```