DATA 643: Project 2

Logan Thomson

6/18/2017

LOAD LIBRARIES

For this project, `recommenderlab` will be utilized mostly; `dplyr` and `ggplot2` are only used for simple data exploration.

```
library(recommenderlab)
library(dplyr)
library(ggplot2)
```

LOAD DATA

After many attempts to use another dataset (Book-Crossings Ratings) without success, we'll utilize the dataset that is pre-loaded with `recommenderlab`.

```
data("MovieLense")
```

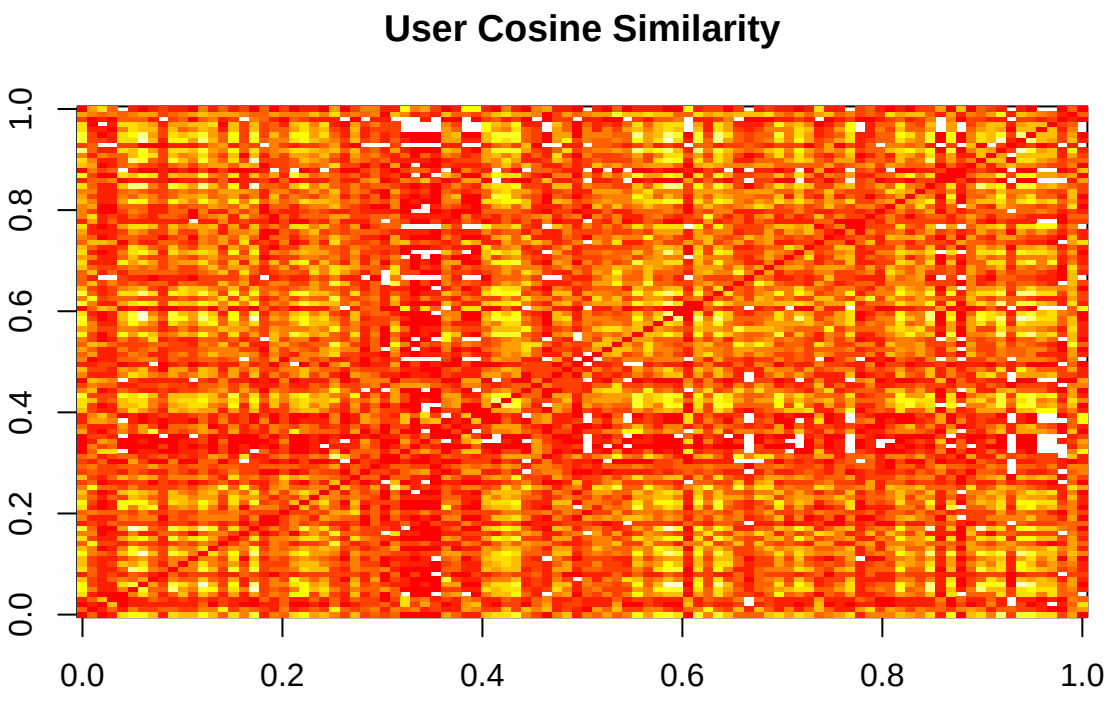
Taking a look at the data, it is stored as the proprietary format for `recommenderlab`, the "realRatingMatrix", which has 943 rows (Users) and 1664 columns (Movies).

```
str(MovieLense)
```

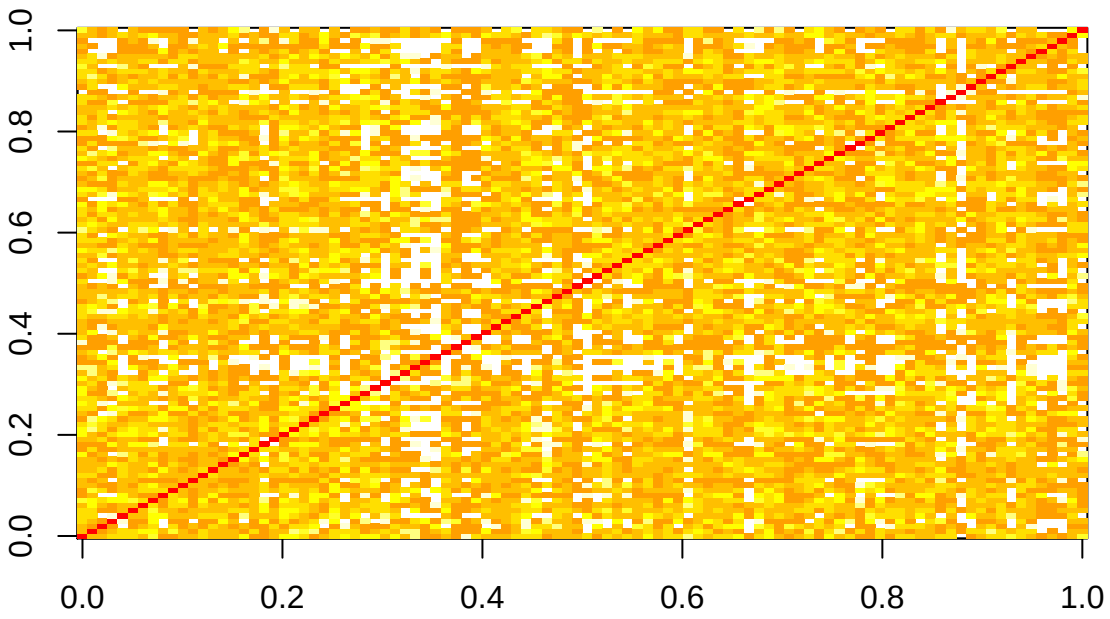
```
## Formal class 'realRatingMatrix' [package "recommenderlab"] with 2 slots
##   ..@ data     :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   .. .. ..@ i       : int [1:99392] 0 1 4 5 9 12 14 15 16 17 ...
##   .. .. ..@ p       : int [1:1665] 0 452 583 673 882 968 994 1386 1605 1904 ...
##   .. .. ..@ Dim     : int [1:2] 943 1664
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : chr [1:943] "1" "2" "3" "4" ...
##   .. .. .. ..$ : chr [1:1664] "Toy Story (1995)" "GoldenEye (1995)" "Four Rooms (1995)" "Get Shorty
##   .. .. ..@ x       : num [1:99392] 5 4 4 4 4 3 1 5 4 5 ...
##   .. .. ..@ factors : list()
##   ..@ normalize: NULL
```

Exploring the data a bit, we can visualize the similarity of users and items using the `similarity` function in Recommenderlab. The differences in the results are due to the way Cosine similarity and Pearson Correlation are calculated. The two plots below show the user-to-user similarity for the first 100 users in the dataset:

```
users_sim_cos <- as.matrix(similarity(MovieLense[1:100, ], method = "cosine", which = "users"))
image(users_sim_cos, main = "User Cosine Similarity")
```

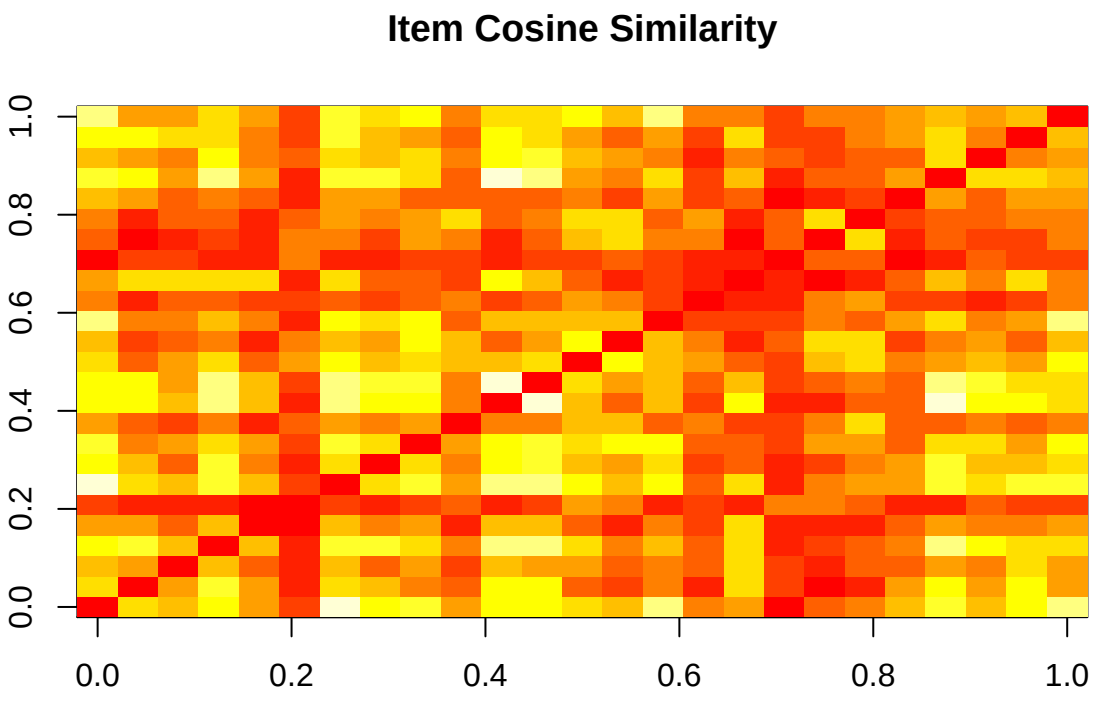



```
users_sim_pson <- as.matrix(similarity(MovieLense[1:100, ], method = "pearson", which = "users"))
image(users_sim_pson, main = "User Pearson Similarity")
```

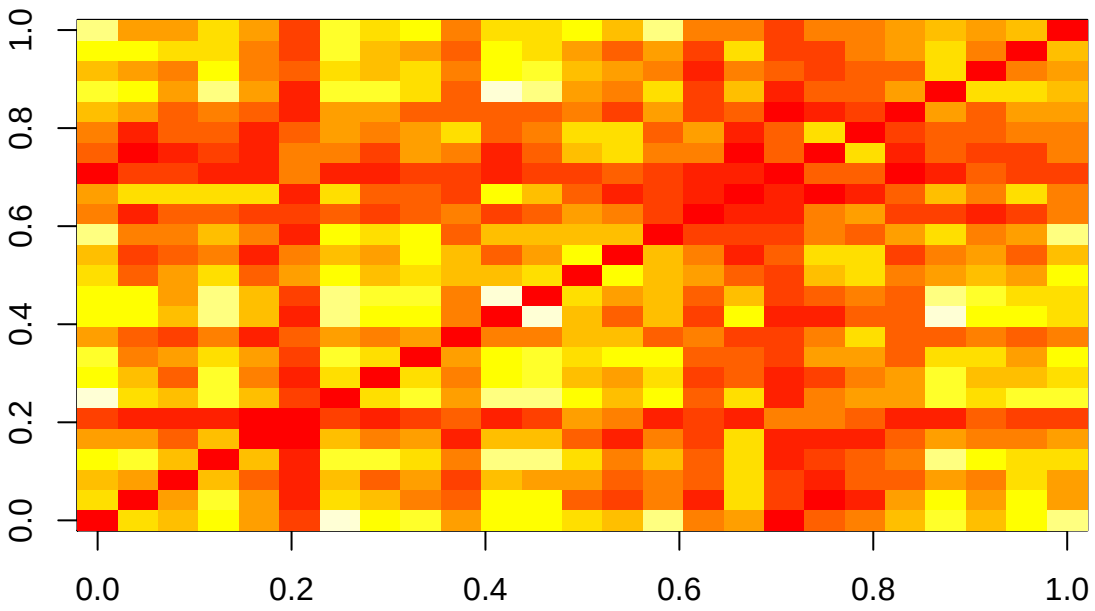


Switching to columns, we can visualize the similarity between movies by their ratings. This time, the plots look much more similar, despite the two different methods being used.

```
item_sim_cos <- as.matrix(similarity(MovieLense[ , 1:25 ], method = "cosine", which = "items"))
image(item_sim_cos, main = "Item Cosine Similarity")
```

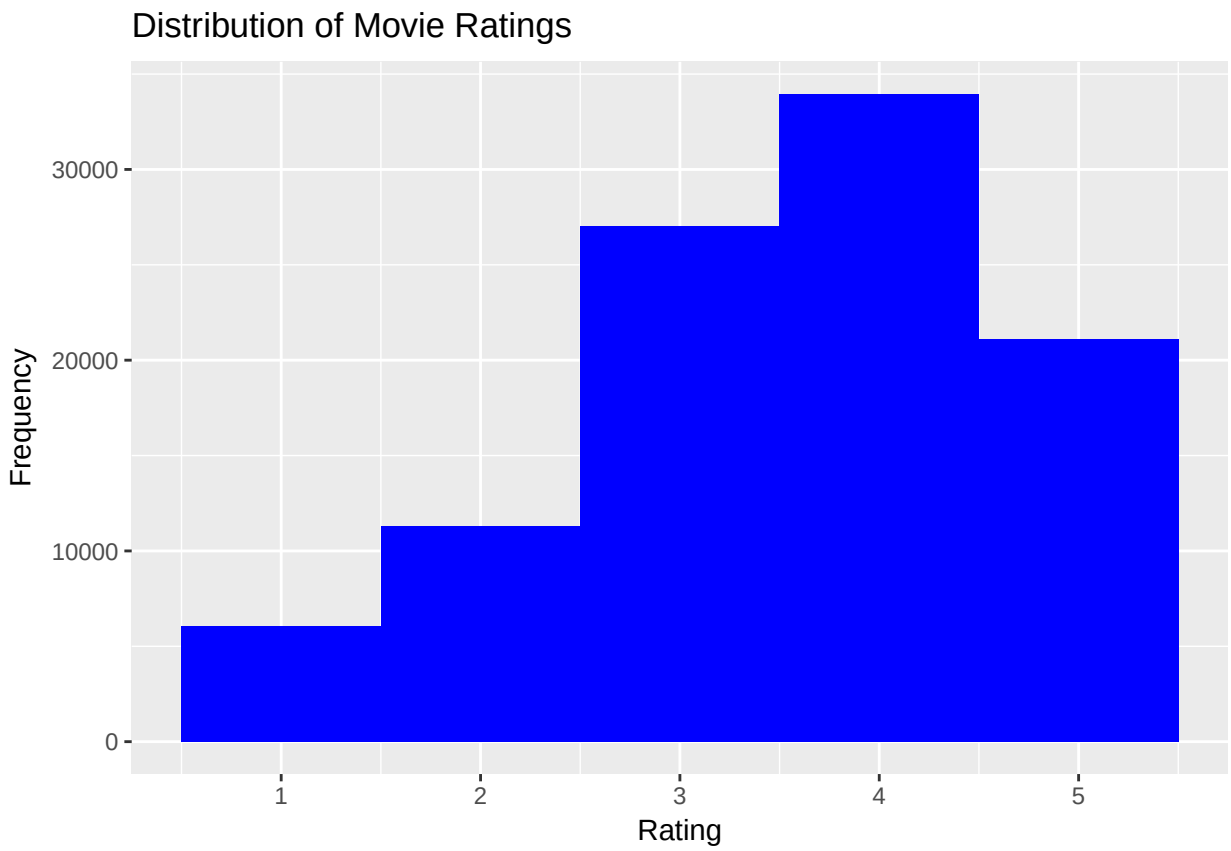



```
item_sim_pson <- as.matrix(similarity(MovieLense[ , 1:25 ], method = "pearson", which = "items"))
image(item_sim_cos, main = "Item Pearson Similarity")
```



We can also examine the distribution of the ratings, which are on a scale of 1 to 5. 0-ratings are given for movies where the user has not given a rating, and account for the majority of the user-item combinations. Below is a histogram of scores, we can see that most are at least a 3, with many users tending to rate things positively. This tends to be the trend with a lot of ratings systems, possibly because we tend to watch movies/tv shows or eat at restaurants we already have an interest in, so we would expect to have a postive reaction to the experience more than not.

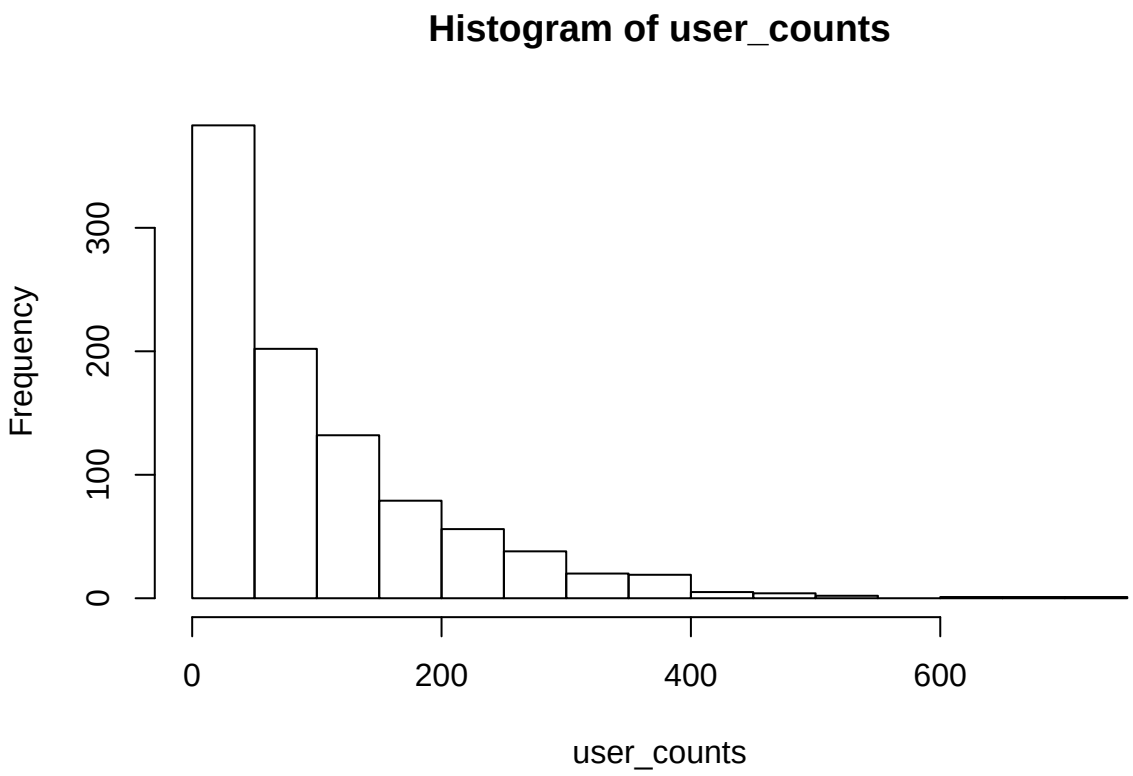
```
ratings <- as.vector(MovieLense@data)
non_zero <- data.frame(ratings[ratings != 0])
colnames(non_zero) <- "rating"

ggplot(non_zero, aes(rating)) + geom_histogram(stat="bin", fill="blue", bins=5) + xlab("Rating") + ylab
```



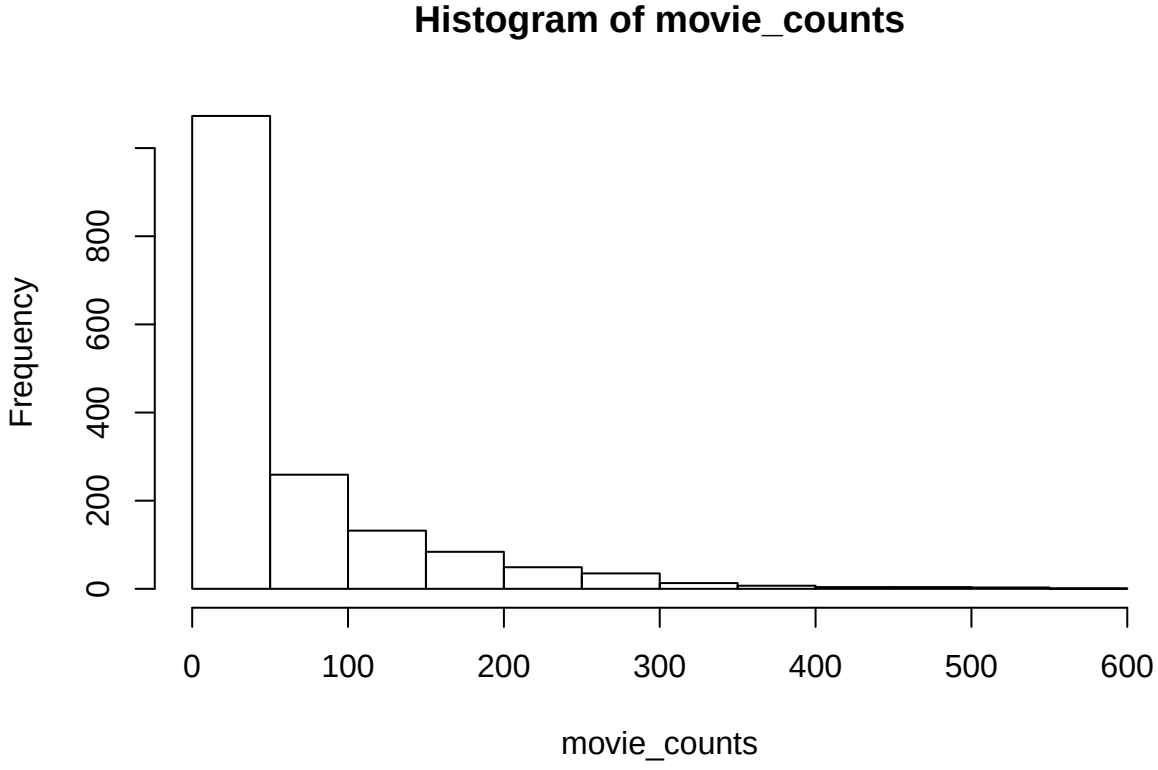
DIVIDING THE DATA

By using `recommenderlab`'s `rowCounts` and `colCounts` functions, we can take a look at the distribution of how many users have submitted ratings, and how many movies have received ratings.

```
user_counts <- rowCounts(MovieLense)
hist(user_counts)
```




```
movie_counts <- colCounts(MovieLense)
hist(movie_counts)
```



As expected, both of these distributions are right-skewed, meaning that most users rate a few movies (less than 100), and most movies have just a few ratings (less than 50). To decrease the sparsity of the data, we'll

work with just the users who have reviewed at least 25 movies, and movies with at least 100 ratings:

```
dense_ratings <- MovieLense[rowCounts(MovieLense) > 25, colCounts(MovieLense) > 100]
dense_ratings
```

```
## 799 x 332 rating matrix of class 'realRatingMatrix' with 61637 ratings.
```

Training & Test Sets

Using `recommenderlab`'s `evaluationScheme` function, we can easily split up the data into training and test sets, rather than using a for-loop or other custom function. The object below is created using the function, and indicates 80% of the data will be put into the training set.

```
evals <- evaluationScheme(dense_ratings, method = "split", train=0.8, given = 9, goodRating = 3, k=1)
```

User-Based Collaborative Filtering

For user-based collaborative filtering, users are compared for similarity, and then the top rated items by the most similar users will be recommended.

Choosing between the two different recommendation algorithms in `recommenderlab` is as simple as calling the different methods in the `Recommender` function. Below the function is called, using `getData` to pull the training information from the evaluation scheme set up above.

Different parameters can be passed to `param`, including normalizing methods ("center" or "Z-score"), similarity methods, nn (no. of similar users), and sampling.

For starters, we'll leave the parameters blank, and just use the "base" `Recommender` function:

```
ubcf_recc <- Recommender(data=getData(evals, "train"), method="UBCF", param=NULL)
ubcf_recc
```

```
## Recommender of type 'UBCF' for 'realRatingMatrix'
## learned using 639 users.
```

After creating the recommender object for the UBCF algorithm, we can call the `predict` function, passing the object created above, and using the test data, accessed from the same eval object created previously (this time passing in "known" instead of "train").

```
ubcf_predict <- predict(object=ubcf_recc, newdata=getData(evals, "known"), n=5, type="ratings")
```

Below are the predicted ratings for the first five users and first five items (movies). We can see that user 13 tends to rate a little more positively, while user 23 skews a little more negative.

```
as(ubcf_predict, "matrix")[1:5, 1:5]
```

```
##    Toy Story (1995) GoldenEye (1995) Get Shorty (1995)
## 1          4.645568         4.550184          4.179560
## 7          4.590820         4.328786          4.379205
## 10         4.386553         4.132372          4.131849
## 11         3.976525         3.584904          3.765917
## 13         4.644922         4.413051          4.449154
##    Twelve Monkeys (1995) Babe (1995)
## 1               4.508677    4.484758
## 7               4.500254    4.465856
## 10              4.212652    4.271152
## 11              3.759144    3.915203
## 13              4.485179    4.647206
```

Item-Based Collaborative Filtering

For item-based filtering, instead of comparing users, we are identifying which movies in the dataset have been rated by the same people, determining their similarity, and then recommending other movies to that user which are similar.

Here, the change in algorithm is as simple as changing the `method` paramter to "IBCF". A similarity method has also been passed to the `parameter`, uh, parameter.

```
ibcf_recc <- Recommender(data=getData(evals, "train"), method="IBCF", parameter=list(method="Jaccard"))
```

Again, the `predict` function is used on the test set, with `n` set to 5 to return the top 5 movie recommendations per user. There are 160 users in the test set (20% of 799 users in `dense_ratings`), so we will return a 160 x 5 result.

```
ibcf_predict <- predict(object = ibcf_recc, newdata=getData(evals, "known"), n=5)
```

By converting the "topNList" to a matrix, we can see the movies recommended for each user.

```
#as(ibcf_predict, "matrix")[1:5, 1:5]
ibcf_matrix <- sapply(ibcf_predict@items, function(x){colnames(dense_ratings)[x]})
ibcf_matrix[1:5]
```

```
## $`1`
## [1] "GoldenEye (1995)"           "Get Shorty (1995)"
## [3] "Twelve Monkeys (1995)"      "Braveheart (1995)"
## [5] "Rumble in the Bronx (1995)"
##
## $`7`
## [1] "Postino, Il (1994)"            "Taxi Driver (1976)"
## [3] "Hudsucker Proxy, The (1994)"   "Much Ado About Nothing (1993)"
## [5] "Lone Star (1996)"
##
## $`10`
## [1] "Birdcage, The (1996)"      "Brazil (1985)"
## [3] "Right Stuff, The (1983)"   "Young Frankenstein (1974)"
## [5] "M*A*S*H (1970)"
##
## $`11`
## [1] "Toy Story (1995)"           "GoldenEye (1995)"
## [3] "Twelve Monkeys (1995)"      "Dead Man Walking (1995)"
## [5] "Usual Suspects, The (1995)"
##
## $`13`
## [1] "Toy Story (1995)"        "Twelve Monkeys (1995)"
## [3] "Babe (1995)"             "Dead Man Walking (1995)"
## [5] "Mighty Aphrodite (1995)"
```

CONCLUSION

`recommenderlab` makes it easy to apply algorithms for user- and item-based collaborative filters, as long as the proprietary data type ('realRatingMatrix') is set up properly. This proved to be a challenge with a different data set, but I will continue investigating how to get this to work properly. While the `Recommend` and `Predict` functions make it easy to perform complex comparisons of data and create recommendations quickly, I also need to become more familiar with the different ways of evaluating the results, and pulling this

information from the data classes in order to create a more useful comparison, rather than just changing the parameters (normalizing or similarity measures) and visually checking the results.