Project 3

Alejandro D. Osborne

June 28, 2019

```r
library(dplyr)
library(tidyr)
library(ggplot2)
library(recommenderlab)
library(reshape2)
library(knitr)
```

The MovieLens matrix was created combining two datasets (movies and ratings). The movies dataset included the movieId, the title and the genres, while the ratings dataset included the userId, the movieId, the movie rating and a timestamp.

```r
movies = read.csv("https://raw.githubusercontent.com/AlejandroOsborne/DATA612/master/movies.csv",
                   header = TRUE, sep =",", stringsAsFactors = FALSE)
ratings = read.csv("https://raw.githubusercontent.com/AlejandroOsborne/DATA612/master/ratings.csv",
                    header = TRUE, sep =",", stringsAsFactors = FALSE)
ratings <- ratings[,c(1,2,3)]
kable(head(movies))
```

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action|Crime|Thriller

```r
kable(head(ratings))
```

userId	movieId	rating
1	1	4
1	3	4
1	6	4
1	47	5
1	50	5
1	70	3

When the rating and movies dataframes were combined, the new dataframe (from which the matrix will be formed) was created. This dataframe contained only the title, the userId and the rating.

```r
movnrate = merge(movies, ratings, by = "movieId")
new = subset(movnrate, select = c("title", "userId", "rating"))
new = unique(new)
kable(head(new))
```

title	userId	rating
Toy Story (1995)	1	4.0
Toy Story (1995)	555	4.0
Toy Story (1995)	232	3.5
Toy Story (1995)	590	4.0
Toy Story (1995)	601	4.0
Toy Story (1995)	179	4.0

```
newmatrix = acast(new, userId~title, value.var="rating", fun=sum)
```

One thing to note, because of R's acast() function (the function that turns the dataframe into a matrix), all missing values were automatically turned into zeroes.In order to avoid overloading R while also avoiding the filtering method, a random sample of both users and movies was taken to create the downsized matrix. This matrix included 200 users and 3000 movies.

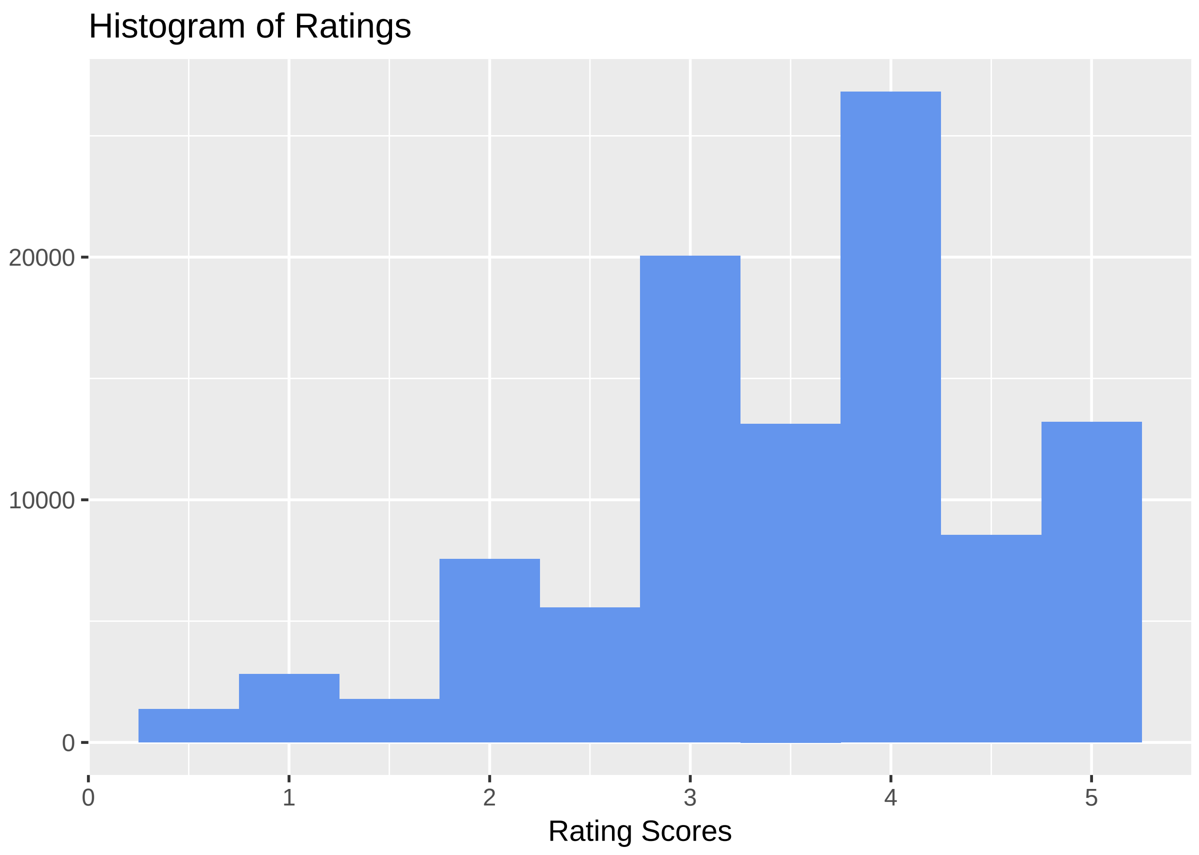
```
premat = as(newmatrix, "realRatingMatrix")
data = premat[sample(610,200), sample(9719, 3000)]
data
```

```
## 200 x 3000 rating matrix of class 'realRatingMatrix' with 600000 ratings.
```

Data Exploration

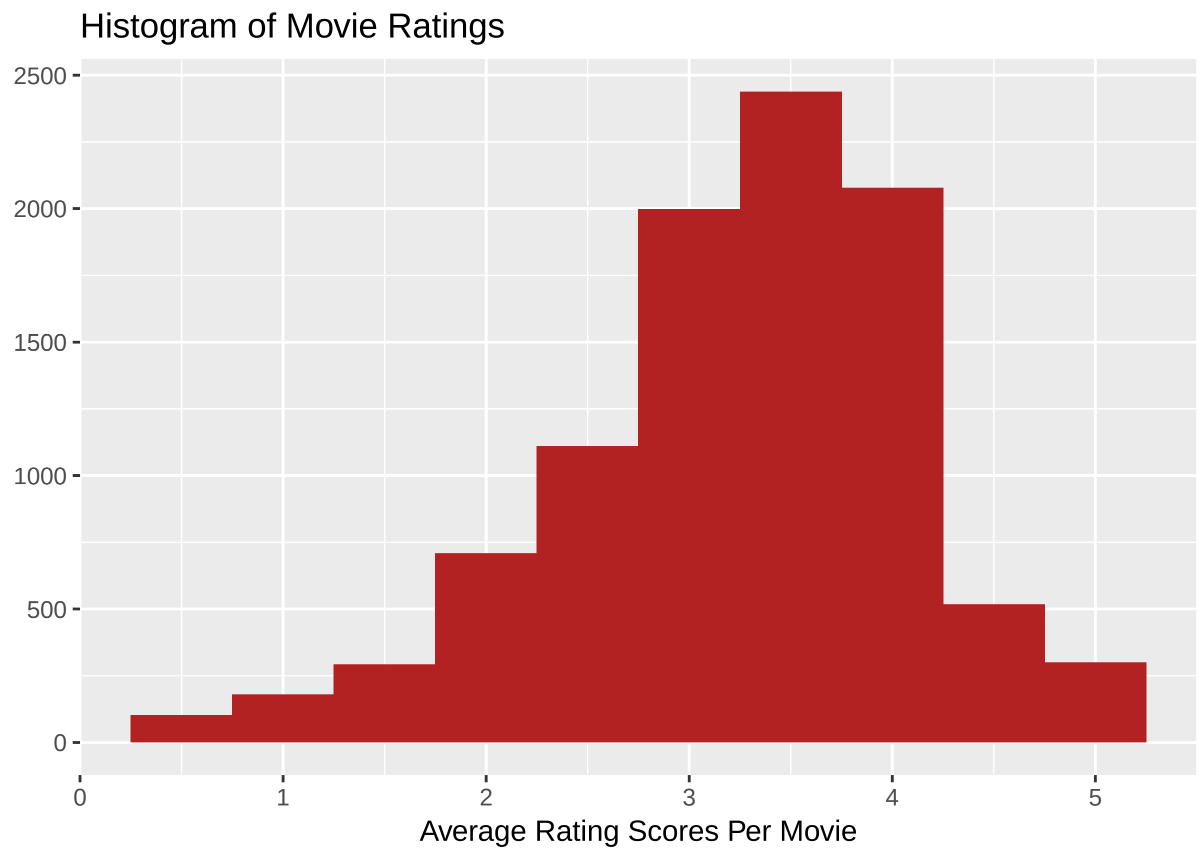
Plot of the ratings distributions will clarify a few things for us, one of those things being what makes a good rating. Looking at the overall ratings, the majority of them were 4 stars, with the most of the votes around the 3-5 star range.

```
# Ratings
qplot(new$rating, geom="histogram", main = "Histogram of Ratings", xlab = "Rating Scores", binwidth = 0
```



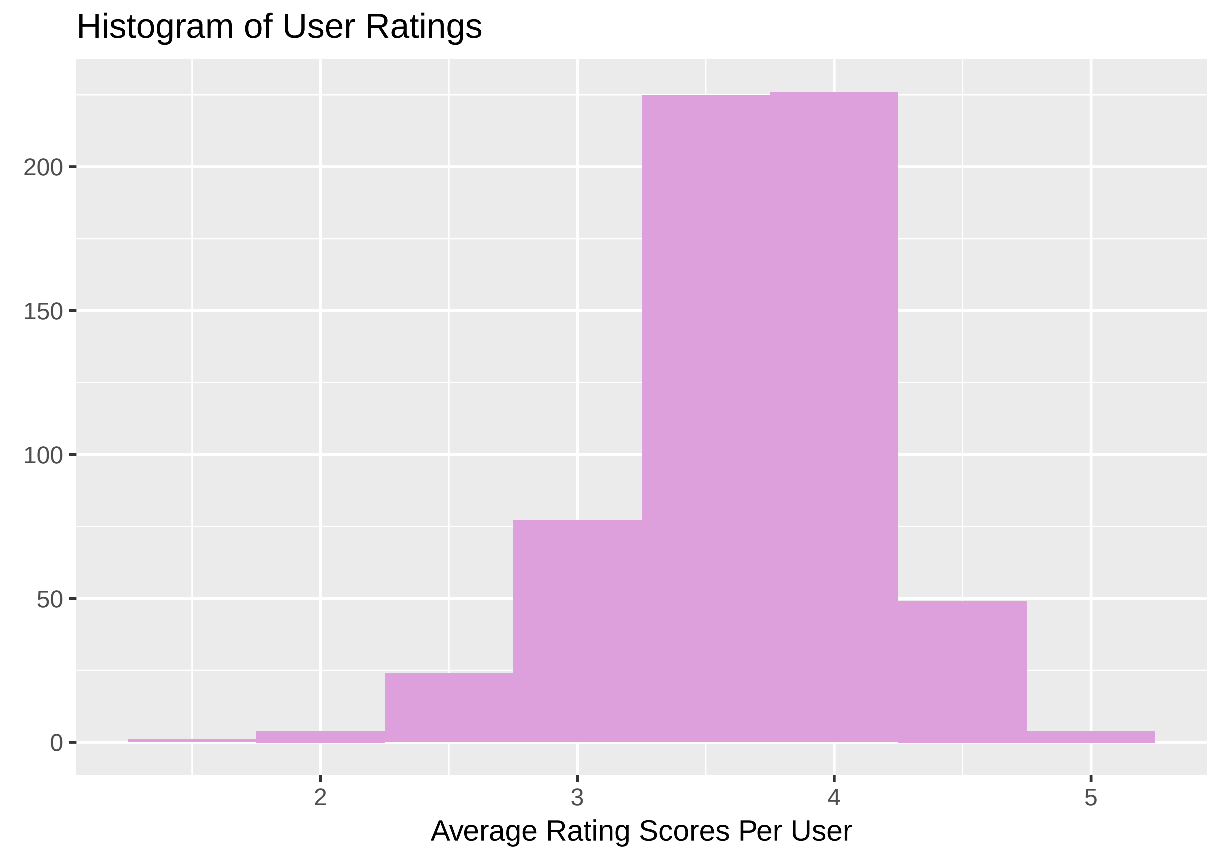
Looking at the average score per movie, the majority of the ratings were 3.5, with less emphasis around the 4.5-5 rating than previously shown. The majority were rated between as 3-4 stars.

```
# Ratings per Movie
new2 = new %>% group_by(title) %>%
  summarise(count = mean(rating))
qplot(new2$count, geom="histogram", main = "Histogram of Movie Ratings", xlab = "Average Rating Scores 
```



If we look at the average rating each of each user, the majority lie0 in the 3.5-4 star range, with more users granting 3 stars than 4.5 stars.

```
# Ratings per User
new3 = new %>% group_by(userId) %>%
  summarise(count = mean(rating))
qplot(new3$count, geom="histogram", main = "Histogram of User Ratings", xlab = "Average Rating Scores Pe
```



After looking at the plots, the data was split into training and test sets (sizes of 80% and 20% respectively).

```
evaluation = evaluationScheme(data, method="split", train=0.8, given=10, goodRating=3.5)
#Evaluation datasets
evAL_train = getData(evaluation, "train")
evAL_known = getData(evaluation, "known")
evAL_unknown = getData(evaluation, "unknown")
```

SVD

The Singlular Value Decomposition (SVD) of a matrix A is the factorization of A into the product of three matrices so that $A = UDV^T$. Here, matrices U and V are orthonormal and matrix D is a diagonal with positive real values. To give an idea of the mentality behind this approach, a random sparse matrix was created.

```
example = as.matrix(data.frame(c(9,7,1,8), c(3,3,8,6), c(1,5,4,2)))
example
```

```
##      c.9..7..1..8. c.3..3..8..6. c.1..5..4..2.
## [1,]             9             3             1
## [2,]             7             3             5
## [3,]             1             8             4
## [4,]             8             6             2
```

Performing a Singular Value Decomposition creates the three U, V and D matrices. The D matrix tells us that the third variable has less strength that the first and second, so it can be set to zero and effectively removed from the U and V matrices. This in turn reduces the original matrice's size without signifcantly affecting the results.

```r
svd(example)
```

```
## $d
## [1] 17.267602  7.119094  3.185659
##
## $u
##            [,1]        [,2]       [,3]
## [1,] -0.5135847  0.47686115 -0.2854817
## [2,] -0.5016047  0.10984955  0.8519934
## [3,] -0.3776426 -0.87069567 -0.0733373
## [4,] -0.5848158  0.04924997 -0.4326997
##
## $v
##            [,1]       [,2]        [,3]
## [1,] -0.7638388  0.6439024 -0.04404608
## [2,] -0.5545416 -0.6896852 -0.46563716
## [3,] -0.3302028 -0.3312463  0.88387894
```

Considering this approach, the SVD method in the recommenderlab package was used on the training set to get in predicted ratings for users and movies. A sample of this is below.

```r
svd_train = Recommender(evAL_train, "SVD")
svd_preds = predict(svd_train, evAL_known, type = "ratings")
getRatingMatrix(svd_preds[c(10,17,1,27,22),5:10])
```

```
## 5 x 6 sparse Matrix of class "dgCMatrix"
##     The Boy and the Beast (2015) Max Keeble's Big Move (2001)
## 521                    0.3994937                    0.3994937
## 560                    0.0000000                    0.0000000
## 5                      0.0000000                    0.0000000
## 236                    0.0000000                    0.0000000
## 148                    0.0000000                    0.0000000
##     Dead Silence (2007) Yours, Mine and Ours (1968) Hyena Road
## 521           0.3956675                   0.3994937  0.4036784
## 560           0.0000000                   0.0000000  0.0000000
## 5             0.0000000                   0.0000000  0.0000000
## 236           0.0000000                   0.0000000  0.0000000
## 148           0.0000000                   0.0000000  0.0000000
##     Life Is Beautiful (La Vita Ã¨ bella) (1997)
## 521                                    0.4008269
## 560                                    0.0000000
## 5                                      0.0000000
## 236                                    0.0000000
## 148                                    0.0000000
```

Comparison

In the previous assignment that used the same dataset, the results showed that the highest performing techniques were the User-to-User Collaborative Filter with Pearson correlation (items were recommended based on similar users) and the Popular method (the most popular items were recommended). Under the assumption reinforced by the plots that the filtering method did not skew the results, the UBCF and Popular methods were compared against the SVD results.

The table below shows that the SVD performed better than the UBCF approach, but was slightly less accurate than the Popular method. This result remains consistent across all values (RMSE, MSE and MAE).

```
# User-User
ubcf_train = Recommender(evAL_train, "UBCF")
ubcf_preds = predict(ubcf_train, evAL_known, type = "ratings")
# Popular
pop_train = Recommender(evAL_train, "POPULAR")
pop_preds = predict(pop_train, evAL_known, type = "ratings")
accuracy = rbind(
  SVD = calcPredictionAccuracy(svd_preds, evAL_unknown),
  UBCF = calcPredictionAccuracy(ubcf_preds, evAL_unknown),
  POPULAR = calcPredictionAccuracy(pop_preds, evAL_unknown)
  )
kable(as.data.frame(accuracy))
```

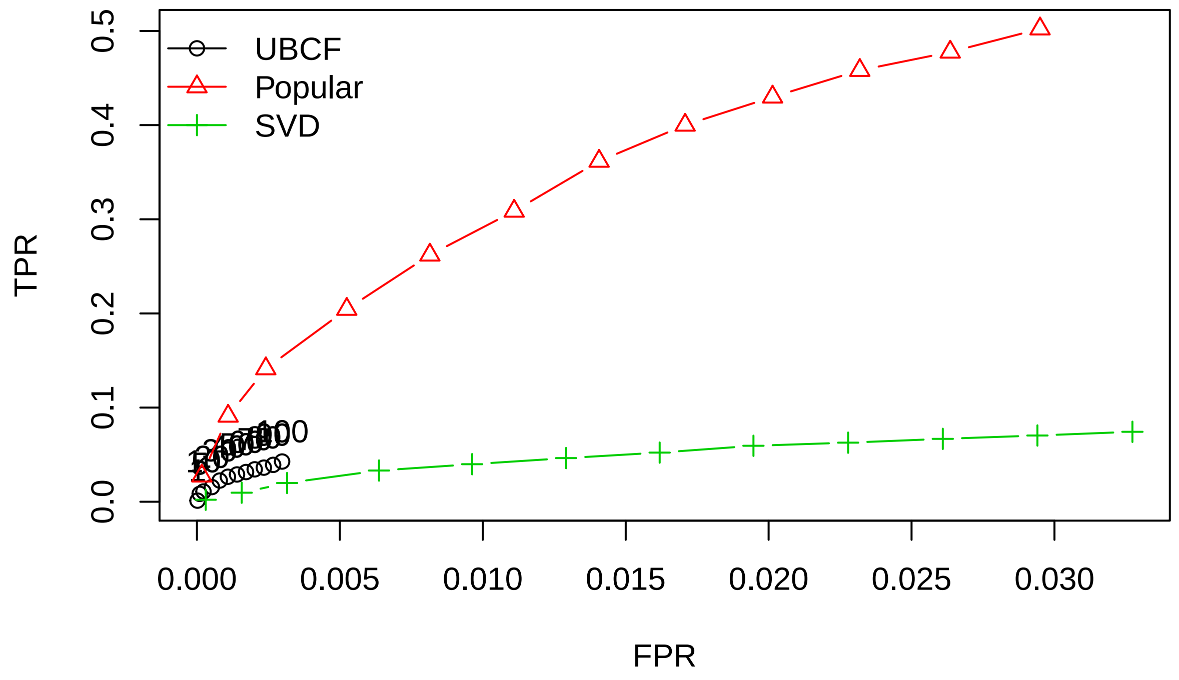
	RMSE	MSE	MAE
SVD	0.4471870	0.1999762	0.1145005
UBCF	0.6383356	0.4074723	0.4463994
POPULAR	0.4292951	0.1842943	0.1687533

The ROC and Precision/Recall plots below show the performance of each of the models.

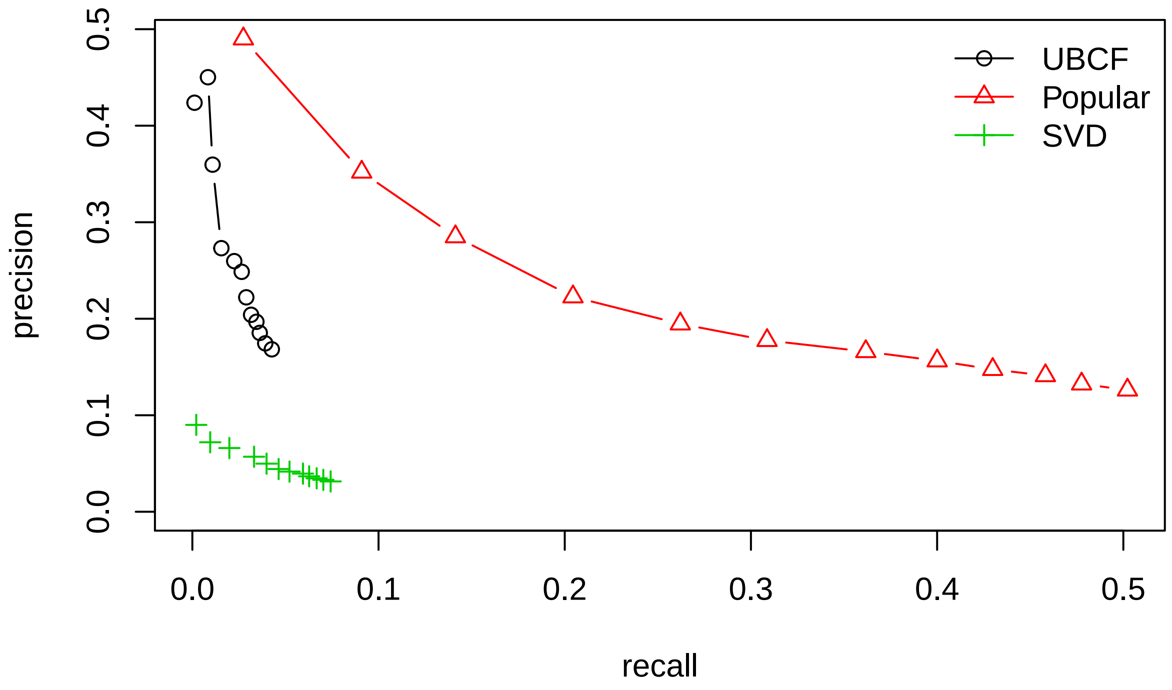
```
eval_sets = evaluationScheme(data = data, method = "cross-validation", k = 4, given = 10, goodRating = 
mult_models = list(
  UBCF = list(name = "UBCF", param = list(method = "pearson")),
  Popular = list(name = "POPULAR", param = NULL),
  SVD = list(name = "SVD", param = NULL)
)
# Testing models
models = evaluate(eval_sets, mult_models, n= c(1, 5, seq(10, 100, 10)))
```

```
## UBCF run fold/sample [model time/prediction time]
##   1  [0.01sec/0.24sec]
##   2  [0.02sec/0.22sec]
##   3  [0.03sec/0.19sec]
##   4  [0.02sec/0.15sec]
## POPULAR run fold/sample [model time/prediction time]
##   1  [0.02sec/0.08sec]
##   2  [0.03sec/0.11sec]
##   3  [0.01sec/0.1sec]
##   4  [0.03sec/0.09sec]
## SVD run fold/sample [model time/prediction time]
##   1  [0.06sec/0.16sec]
##   2  [0.07sec/0.14sec]
##   3  [0.06sec/0.16sec]
##   4  [0.06sec/0.16sec]
```

```
# Plotting models
plot(models, annotate = T, legend="topleft")
```



```r
plot(models, "prec/rec", annotate = F, main="Precision/Recall", legend="topright")
```



Despite what was expected from the accuracy table above, the SVD model did not perform as well as expected compared to the UBCF and Popular model. The Popular model performed the best, and in terms of Precision/Recall (and looking at the ROC curves), the SVD model was significanlty below the Popular model's.

Conclusion

The best approach using this downsized MovieLens dataset was, once again, by recommending Popular movies. That being said, the Singular Value Decomposition approach did not perform poorly. When viewing the results by comparing RMSE, MSE and MAE values, the SVD method performed slightly beneath the Popular one, with the UBCF lagging further behind. Though both the ROC and Precision/Recall plots showed the SVD underperforming (even to the UBCF), the SVD method would be my second choice after the Popular approach based on the results of the accuracy table.