Project 4

Alejandro D. Osborne

July 2, 2019

Data input

The dataset used was the Amazon Digital Music dataset. This dataset was retrieved from the http://jmcauley.ucsd.edu/data/amazon/links.html website using the "ratings only" dataset which includes n oreviews, only (user, item, rating, timestamp) tuples. This dataset includes 478,235 users, 266,414 items (music/albums) and ratings that range from 1 to 5 (least to most liked).

Loading libraries/dataset

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

The table below is a sample of the dataset's format that later was transformed into a ratings matrix. The timestamp column was removed and only the user, item and rating information was retained. The user column does not include names or usernames, but rather an ID, most likely to maintain user privacy, while the item numer/string is the item's product ID.

```r
music1 = read.csv("https://raw.githubusercontent.com/AlejandroOsborne/DATA612/master/Amazon_Music_1.csv
music2 = read.csv("https://raw.githubusercontent.com/AlejandroOsborne/DATA612/master/Amazon_Music_2.csv
music = rbind(music1, music2)
music = music[,1:3]
kable(head(music))
```

user	item	rating
A2EFCYXHNK06IS	5555991584	5
A1WR23ER5HMAA9	5555991584	5
A2IR4Q0GPAFJKW	5555991584	4
A2V0KUVAB9HSYO	5555991584	4
A1J0GL9HCA7ELW	5555991584	5
A3EBHHCZO6V2A4	5555991584	5

Creating Matrix

The matrix was too sparse to retain and evaluate all the ratings, and bacause R's capacity to handle large matrices isn't the best, the matrix was narrowed down to the users and items that received the most traffic. The resulting matrix has around 18,000 ratings.

```r
set.seed(101)
a = data.frame(head(sort(table(music$user), decreasing = T), 1835))
colnames(a) = c("user", "count")
user_merge = merge(music, a, by = "user")
b = data.frame(head(sort(table(user_merge$item), decreasing = T), 988))
```

```
colnames(b) = c("item", "count")
item_merge = merge(user_merge, b, by = "item")
final = subset(item_merge, select = c("user", "item", "rating"))
```

```
data = as(final, "realRatingMatrix")
data = data[rowCounts(data) > 5, colCounts(data) > 5]
data
```

```
## 744 x 988 rating matrix of class 'realRatingMatrix' with 18227 ratings.
```

```
print(paste0("Minimum number of ratings: ", min(rowCounts(data))))
```

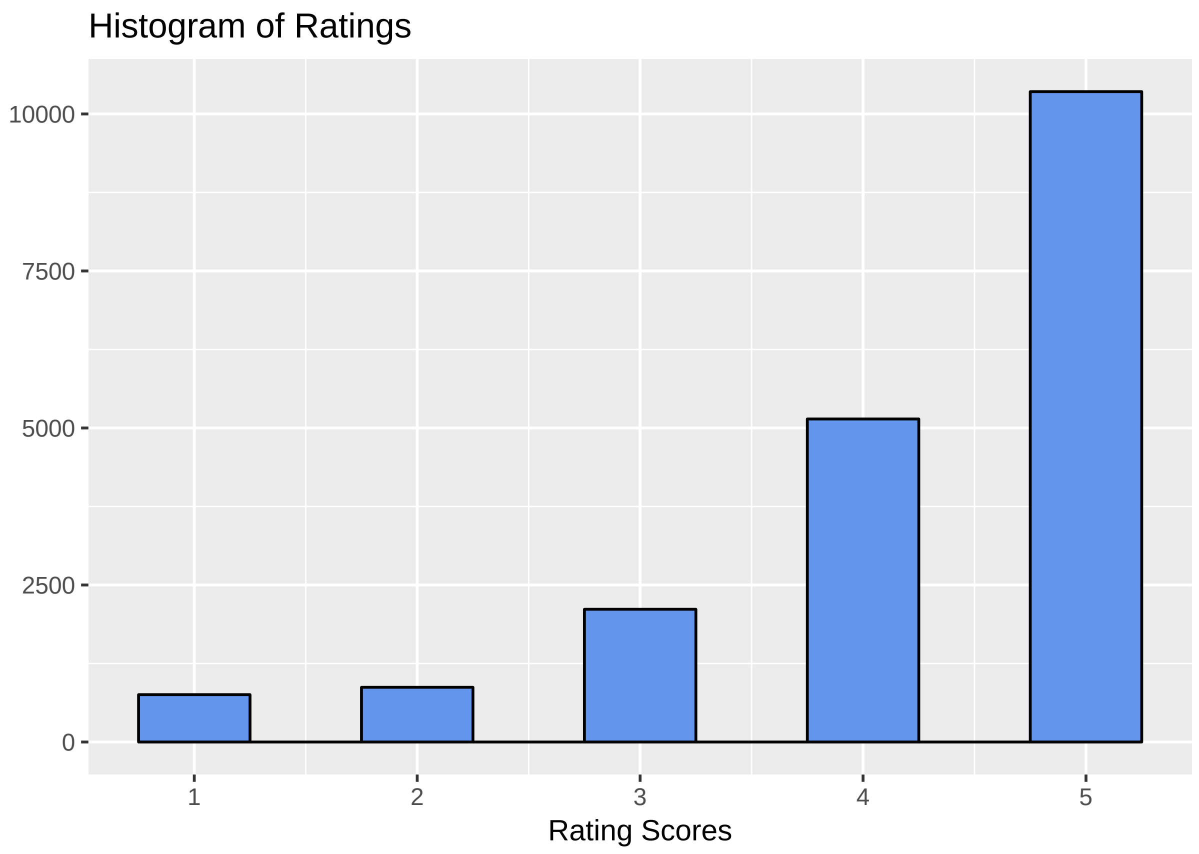
```
## [1] "Minimum number of ratings: 6"
```

Data Exploration

The histograms of the ratings distributions allows us to understand what constitutes a good rating, as well as letting us visually viewing how sparse the matrix is and whether patterns exist within the data.

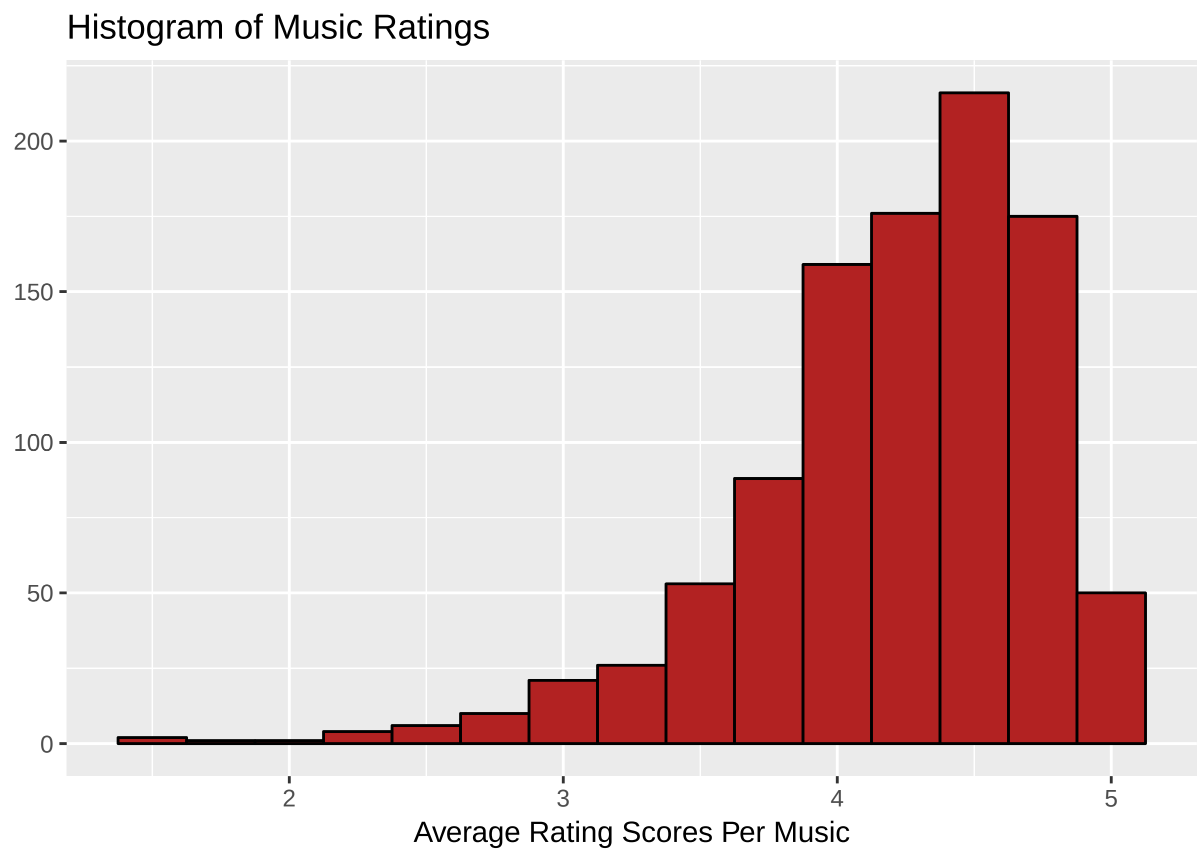
The ratings decrease in frequency from 5 to 1. the number of 1 and 2 ratings may (when added), approximate to the 3-star counts. This will probably show in the other histograms a tendency for users and ratings to have a score around the 4-5 star mark.

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



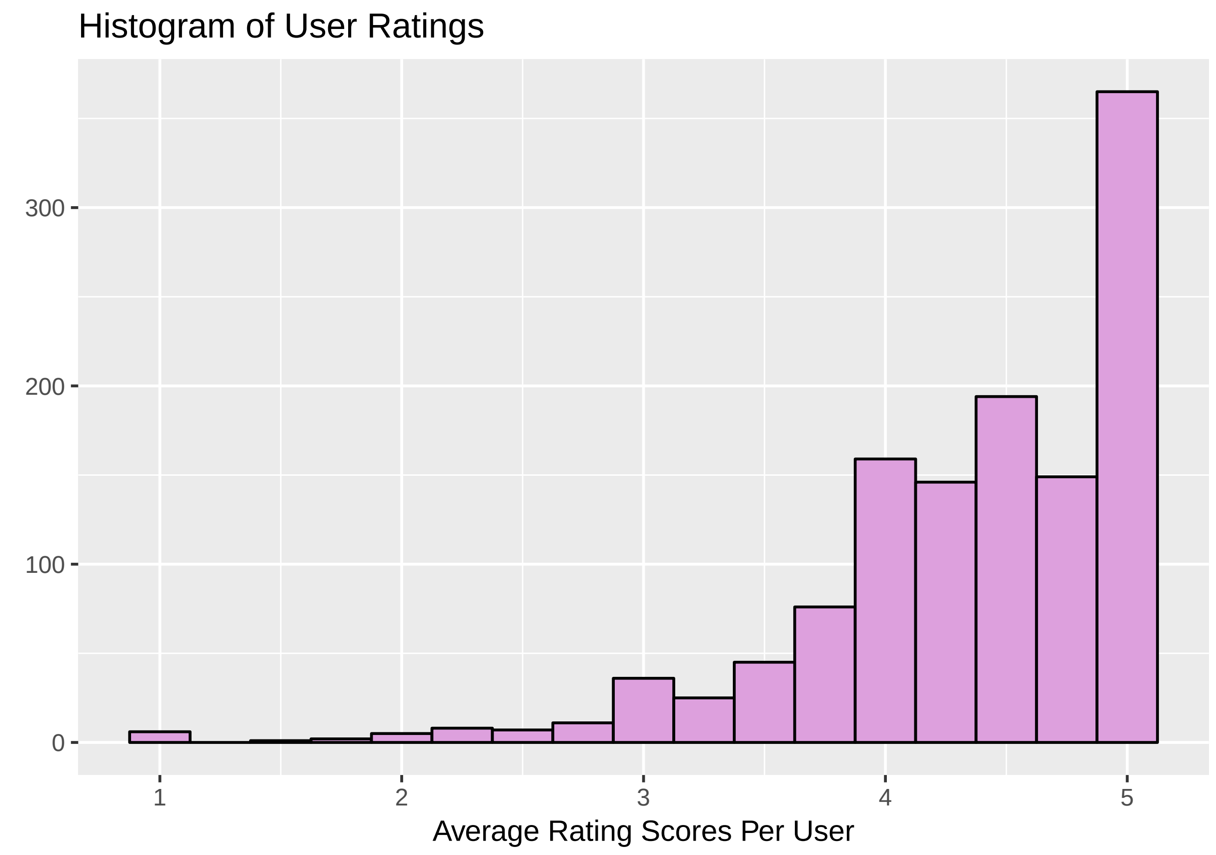
The ratings per music item are primarily around the 4-5 star rating, with the 4.5 rating being the most frequent grade. Few scores below 3 stars are given to any one music item.

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



An observation I found to be strange, very few overall give low scores, in fact, many appear to give 5 stars on average. However, even to this end the mean/median is around the 4.5 mark.

```r
# Ratings per Music
new2 = final %>% group_by(user) %>%
  summarise(count = mean(rating))
qplot(new2$count, geom="histogram", main = "Histogram of User Ratings", xlab = "Average Rating Scores Pe
```



Accuracy

A recommender system was created and its performance was evaluated through accuracy scores (RMSE, MSE and MAE). The data was split into training and test datasets (80 and 20 percent, respectively).

```
evaluation = evaluationScheme(data, method="split", train=0.8, given=5, goodRating=4)
#Evaluation datasets
ev_train = getData(evaluation, "train")
ev_known = getData(evaluation, "known")
ev_unknown = getData(evaluation, "unknown")
```

First Recommender

The primary recommender system was a User Based Collaborative Filter, where recommendations are made by determining the similarity between users. This was determined based on the hunch from the heatmap that there more likely exists a similarity relationship between users than items.

```
# User-User
ubcf_train = Recommender(ev_train, "UBCF")
ubcf_preds = predict(ubcf_train, ev_known, type = "ratings")
ubcf_preds
```

```
## 149 x 988 rating matrix of class 'realRatingMatrix' with 143518 ratings.
```

Other Recommenders

To determine the best recommender system, other algorithms were used for comparison. Those included: an Item Based Collaborative Filter (IBCF), Singular Value Decomposition (SVD) and a Popular algorithm. The results from evaluating these recommender systems were compared through an accuracy table that included the accuracy measurements (RMSE, MSE, MAE).

```r
# Item-Item
ibcf_train = Recommender(ev_train, "IBCF")
ibcf_preds = predict(ibcf_train, ev_known, type = "ratings")
# Popular
pop_train = Recommender(ev_train, "POPULAR")
pop_preds = predict(pop_train, ev_known, type = "ratings")
# SVD
svd_train = Recommender(ev_train, "SVD")
svd_preds = predict(svd_train, ev_known, type = "ratings")
```

Comparison

The results from the accuracy comparison measures, shown in the table below, determined that the Popular model was the most successful, with the UBCF model coming in second (and the SVD model being a close third). The IBCF model underperformed greatly, in comparison.

```r
accuracy = rbind(
  UBCF = calcPredictionAccuracy(ubcf_preds, ev_unknown),
  IBCF = calcPredictionAccuracy(ibcf_preds, ev_unknown),
  SVD = calcPredictionAccuracy(svd_preds, ev_unknown),
  POPULAR = calcPredictionAccuracy(pop_preds, ev_unknown)
  )
acc_df = round(as.data.frame(accuracy), 3)
kable(acc_df[order(acc_df$RMSE),])
```

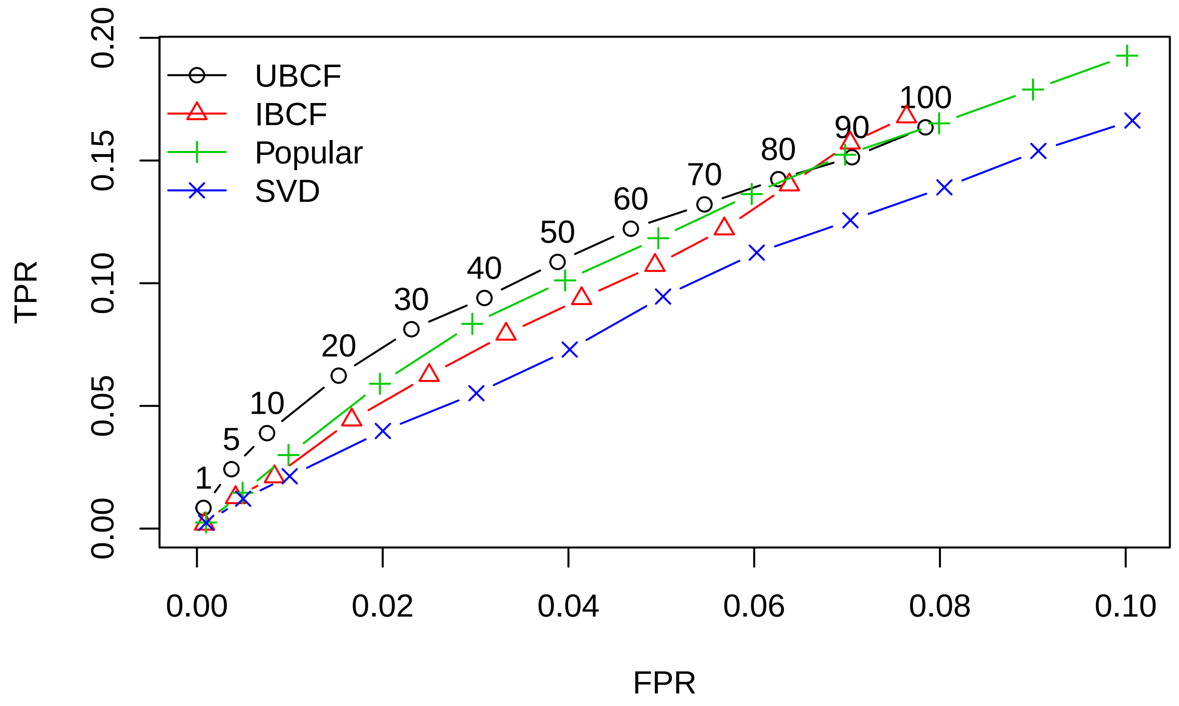
	RMSE	MSE	MAE
POPULAR	1.006	1.011	0.744
UBCF	1.051	1.104	0.764
SVD	1.058	1.119	0.766
IBCF	1.535	2.358	1.011

However, when viewing the ROC plots of the models, the IBCF model did not perform as poorly as assumed in the accuracy table. Around the 0.07 mark on the FPR axis, it overtook the UBCF model and became rather close to the Popular model.

```r
eval_sets = evaluationScheme(data = data, method = "cross-validation", k = 4, given = 5, goodRating = 4
mult_models = list(
  UBCF = list(name = "UBCF", param = list(method = "pearson")),
  IBCF = list(name = "IBCF", 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  [0sec/0.3sec]
##   2  [0sec/0.36sec]
##   3  [0sec/0.3sec]
##   4  [0sec/0.35sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [6.26sec/0.06sec]
##   2  [6.23sec/0.03sec]
##   3  [6.21sec/0.06sec]
##   4  [6.15sec/0.06sec]
## POPULAR run fold/sample [model time/prediction time]
##   1  [0sec/0.3sec]
##   2  [0sec/0.32sec]
##   3  [0sec/0.3sec]
##   4  [0sec/0.29sec]
## SVD run fold/sample [model time/prediction time]
##   1  [0.08sec/0.14sec]
##   2  [0.08sec/0.14sec]
##   3  [0.08sec/0.15sec]
##   4  [0.05sec/0.17sec]
```

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



Increase Serendipity

The problem with determining the success of introducing serendipity in a model is that it requires user satisfaction, something that the ratings alone cannot provide. Originally, the more ideal approach to include serendipity into a model would be to create recommendations for a user, look at the least likely of the most preferable recommendations and include those in a list for the user to view (assuming that the user has not already heard this music). However, since there is no way to measure the accuracy of a serendipitous model using an offline dataset with only ratings, the approach taken was slightly different. This approach aimed to recommend least likely items from the list, which would most likely lead to a sense of "serendipity".

Serendipity

Since the Popular system was the most successful, the most popular recommendentations were extracted. It is visible from the recommendation lists of the five first users that the top 20 recommendations are approximately the same. Variation appears to be introduced more after the first 20.

```
pres = predict(pop_train, 1:100 , data = ev_train, n = 20)
pres@items[1:5]
```

```
## $A2CA36LT9SXGNI
##  [1] 151 475 319  43 369 142 186 362  44  38  42 245 483 652  27 438 781
## [18]  40 370 430
##
## $A1534MBU6VJXYN
##  [1] 375 151 475 319  41  43 369 142 186 107 362  44 442  38  42 245 483
## [18] 652  27 438
##
## $A3O8YT41TDXLOB
##  [1] 375 151 475 319  41  43 369 142 186 107 362  44 442  38  42 245 483
## [18] 652  27 438
##
## $A3GKOMCQTTWPUI
##  [1] 375 151 319  43 369 142 186 107 362  44 442  38  42 245 483 652  27
## [18] 781  40 430
##
## $A1X93ES4DITTWK
##  [1] 375 151 475 319  41  43 369 142 107 362  44 442  38  42 245 483 652
## [18]  27 438 781
```

Since the first 20 values were primarily similar through all users, a user was selected at random and their first 20 music item recommendations were used as the standard to be removed from the matrix so that more "serendipitous" (less popular) recommendations could be made using other algorithms.

```
values = unlist(as.vector(head(sample(pres@items[1:100]), 1)), use.names=FALSE)
values
```

```
##  [1] 375 151 475 319  41  43 369 186 107 362 442  38  42 245 483 652  27
## [18] 438 781  40
```

```
data2 = data[,-values]
evaluation2 = evaluationScheme(data2, method="split", train=0.8, given=5, goodRating=4)
#Evaluation datasets
ev_train2 = getData(evaluation2, "train")
ev_known2 = getData(evaluation2, "known")
ev_unknown2 = getData(evaluation2, "unknown")
# User-User
seren_ubcf_train = Recommender(ev_train2, "UBCF")
seren_ubcf_preds = predict(seren_ubcf_train, ev_known2, type = "ratings")
# Item-Item
seren_ibcf_train = Recommender(ev_train2, "IBCF")
seren_ibcf_preds = predict(seren_ibcf_train, ev_known2, type = "ratings")
# Popular
```

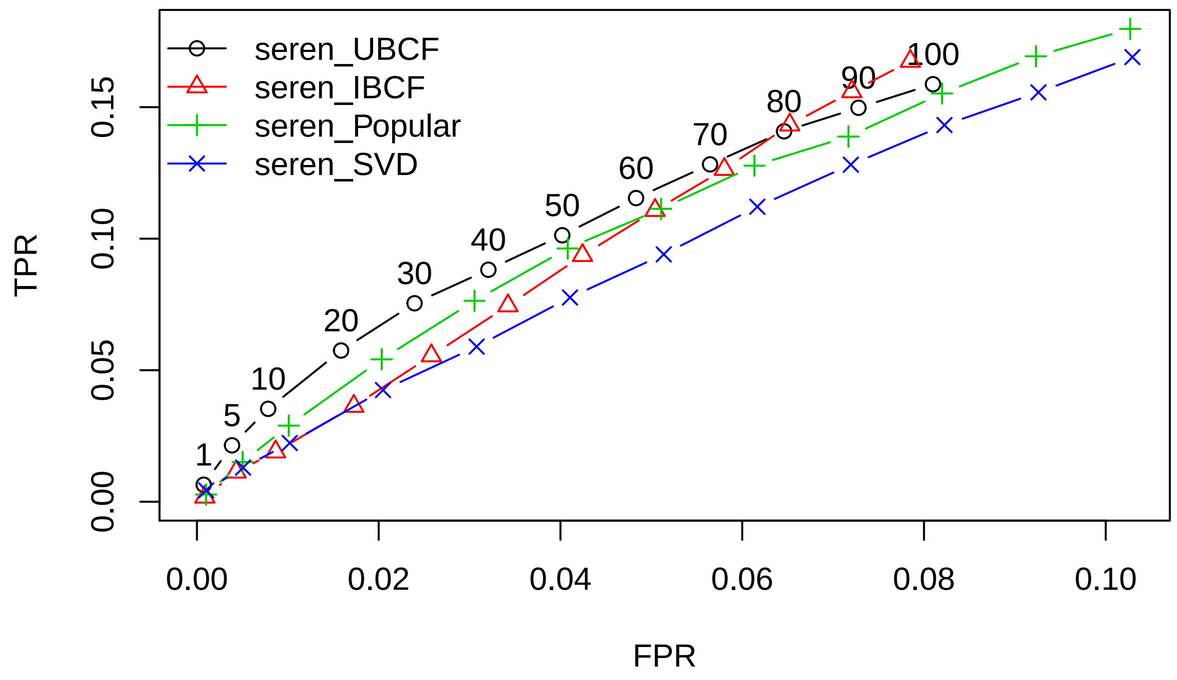
```
seren_pop_train = Recommender(ev_train2, "POPULAR")
seren_pop_preds = predict(seren_pop_train, ev_known2, type = "ratings")
# SVD
seren_svd_train = Recommender(ev_train2, "SVD")
seren_svd_preds = predict(seren_svd_train, ev_known2, type = "ratings")
```

The UBCF, IBCF and SVD algorithms were used with this "less popular" matrix to create recommendations. The UBCF algorithm was also used, but it appeared unlikely to hold any meaningful significance. The ROC plot below shows the UCBF model performing the best until the 0.05 FPR mark, where it is overtaken by the Item Based model. The SVD model underperformed (comparitively) but not by much.

```
eval_sets2 = evaluationScheme(data = data2, method = "cross-validation", k = 4, given = 5, goodRating =
mult_models2 = list(
  seren_UBCF = list(name = "UBCF", param = list(method = "pearson")),
  seren_IBCF = list(name = "IBCF", param = list(method = "pearson")),
  seren_Popular = list(name = "POPULAR", param = NULL),
  seren_SVD = list(name = "SVD", param = NULL)
)
# Testing models
models2 = evaluate(eval_sets2, mult_models2, n= c(1, 5, seq(10, 100, 10)))
```

```
## UBCF run fold/sample [model time/prediction time]
##   1  [0sec/0.28sec]
##   2  [0.02sec/0.26sec]
##   3  [0.02sec/0.3sec]
##   4  [0sec/0.28sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [5.08sec/0.06sec]
##   2  [4.94sec/0.03sec]
##   3  [5.04sec/0.02sec]
##   4  [4.96sec/0.03sec]
## POPULAR run fold/sample [model time/prediction time]
##   1  [0.01sec/0.25sec]
##   2  [0sec/0.3sec]
##   3  [0sec/0.26sec]
##   4  [0sec/0.28sec]
## SVD run fold/sample [model time/prediction time]
##   1  [0.04sec/0.14sec]
##   2  [0.05sec/0.09sec]
##   3  [0.04sec/0.14sec]
##   4  [0.05sec/0.13sec]
```

```
# Plotting models
plot(models2, annotate = T, legend="topleft")
```



Accuracy Comparisons

When comparing the "serendipitous" models, they performed better than their counterparts. The IBCF model performed below the regular Popular, UBCF and SVD models. Since the "serendipitous" Popular model's results cannot really be trusted, it is probably safest to consider the best model as the "regular" Popular algorithm.

```
accuracy2 = rbind(
  seren_UBCF = calcPredictionAccuracy(seren_ubcf_preds, ev_unknown2),
  seren_IBCF = calcPredictionAccuracy(seren_ibcf_preds, ev_unknown2),
  seren_SVD = calcPredictionAccuracy(seren_svd_preds, ev_unknown2),
  seren_POPULAR = calcPredictionAccuracy(seren_pop_preds, ev_unknown2)
  )
acc_df2 = round(as.data.frame(accuracy2), 3)
comp = rbind(acc_df, acc_df2)
kable(comp[order(comp$RMSE),])
```

	RMSE	MSE	MAE
seren_POPULAR	0.928	0.861	0.690
seren_UBCF	0.974	0.948	0.712
seren_SVD	0.988	0.975	0.721
POPULAR	1.006	1.011	0.744
UBCF	1.051	1.104	0.764
SVD	1.058	1.119	0.766
seren_IBCF	1.379	1.900	0.923
IBCF	1.535	2.358	1.011

	RMSE	MSE	MAE

Conclusion

To best determine the serendipity of a model, it is best to have a serendipity rating (a rating that expresses surpise on a negative to positve scale, with zero probably meaning no surprpise). Overall, if ignoring the popular model, a user-to-user algorithm is the most successful with this dataset, though removing the 20 most popular music items lead to a lower RMSE value.

One way to determine serendipity (or recommendation success) using an online environment is by tracking whether a recommended album/track is played by a user (or by seeing if a user explores music by a recommended artist). This may include calculating a shift in music preferences (by using a diversity metric to see if the user's taste has expanded into something they hadn't tried before). This could either be recorded by retrieving how many times a user clicks on a newly recommended artist/album or by measuring the number of times/overall time a track has been listened to.