Project 3 Matrix Factorization Method - SVD

Cheryl Bowersox

June 26, 2018

This project uses the recommenderlab package to create recommendations using the Beer Advocate data set used in previous projects. The recommendations are created using the SVD function from the recommenderlab package to reduce the size of the data, and the model is evaluated for accuracy and predictive power.

Description of source data: Unique Users(users) are defined by variable review_profilename Unique beer (items) are defined by variable beer_beerid Several beer attributes are rated within this data set, but for the purposes of this project only the overall rating, given by 'review_overall' will be used.

Data Source:

https://data.world/socialmediadata/beeradvocate (https://data.world/socialmediadata/beeradvocate)

Data Exploration

The total number of distinct beers in the data is 42719 and the beers cover a wide range of number of ratings, from as few as 1 to 3000. The beer data is sparse, as there are 41946 beers that have received less than 300 ratings. This accounts for0.981905 of the total number of items.

The number of ratings per user shows a fairly uniform distribution across the 28762 distinct users, with the number of ratings ranging from 1 to 3763. There are 26411 users that have rated less than 100 items. This accounts for 0.9182602 of the total number of users.

Our raw data contains 1048575 data points, and a ratings matrix for this data would be 28762 X 42719. By removing beers that have received less than 300 ratings and users that have rated less than 100 of the remaining beers we can reduce the size significantly.

We still retain valuable information for beers that have frequent ratings and can create predictions for users with longer history of ratings, but this method is also limited. It lends itself to reinforcing existing options, without offering opportunity for new items to be recommended. It also does not allow for an intelligent way to recommend for new users. One way this can be addressed in a full model is to randomly recommend to well-known users a sample of beers excluded from the original model but that have been rated positively or neutrally. This creates an opportunity to gain new information on both the beers and the users. For newer users excluded from our model due to lack of ratings, we can recommend a random selection of positively rated beers to develop enough history to be included in the model.

```
#keep only items with more than than 100 ratings, and of those only users rating more
than 100.
v <- sparseratings%>% filter(count > 300)
#filter for items in list v
dfbeerrm <- beerdata%>%
  filter(beer_beerid %in% v$beer_beerid)


w <- dfbeerrm%>%select(beer_beerid, review_profilename)%>%
  group_by(review_profilename)%>%
  summarise(count=n())%>% filter(count > 100)
#filter for users in list w
dfbeerrm <- dfbeerrm%>%
  filter(review_profilename %in% w$review_profilename)

items <- dfbeerrm%>%select(beer_name, review_overall)%>%
  group_by(beer_name)%>%
  summarise(count=n())%>%arrange(-count)
items_n <- nrow(items)

users<- dfbeerrm%>%select(review_profilename, review_overall)%>%
  group_by(review_profilename)%>%
  summarise(count=n())%>%arrange(-count)


users_n <- nrow(users)


df <- dfbeerrm%>%select(user = review_profilename,beer =beer_name,rate =review_overal
l)
df[is.na(df)] <- 0

#Head of data

(head(df))
```

```
##         user         beer rate
## 1     jdhilt Amstel Light  2.5
## 2      Brent Amstel Light  3.0
## 3   Mora2000  Caldera IPA  4.0
## 4    Rutager  Caldera IPA  4.0
## 5 beerman207  Caldera IPA  4.5
## 6     JayQue  Caldera IPA  4.0
```

The new data set containing only users that have rated more than 100 beers, and only beers with more than 300 ratings has been reduced to 289657 rows, with 769 items and 1401 users. By removing these items we have reduced the ratings matrix to a more manageable size.

Now that we have a reduced data set we will set up a training and testing proceedure using the recommenderlab package function evaluation scheme.

In this scheme I chose to split the data into 80% training and 20% testing, a positive rating is considered 4 and above, with all but 10 ratings given for the testing set, to evaluate how the model predicts the unknown items in the testing set.

```
# create ratings matrix
ratemat <- as(df,"realRatingMatrix")
#norm_ratemat <- normalize(ratemat)


#number to take as given in testing
numgiven <- -10 # min number of ratings is 20
splitsville <- .20 #amount to keep as testing data
posthreshold <- 4 #what do we consider a positive rating
evals <- 4  ## number of runs to evalualte



#split data into train and test

evalsplit <- evaluationScheme(data=ratemat, method = "split", train = splitsville,
                              given =numgiven, goodRating = posthreshold, k = evals)
(evalsplit)
```

```
## Evaluation scheme using all-but-10 items
## Method: 'split' with 4 run(s).
## Training set proportion: 0.200
## Good ratings: >=4.000000
## Data set: 1401 x 769 rating matrix of class 'realRatingMatrix' with 284239 ratings.
```

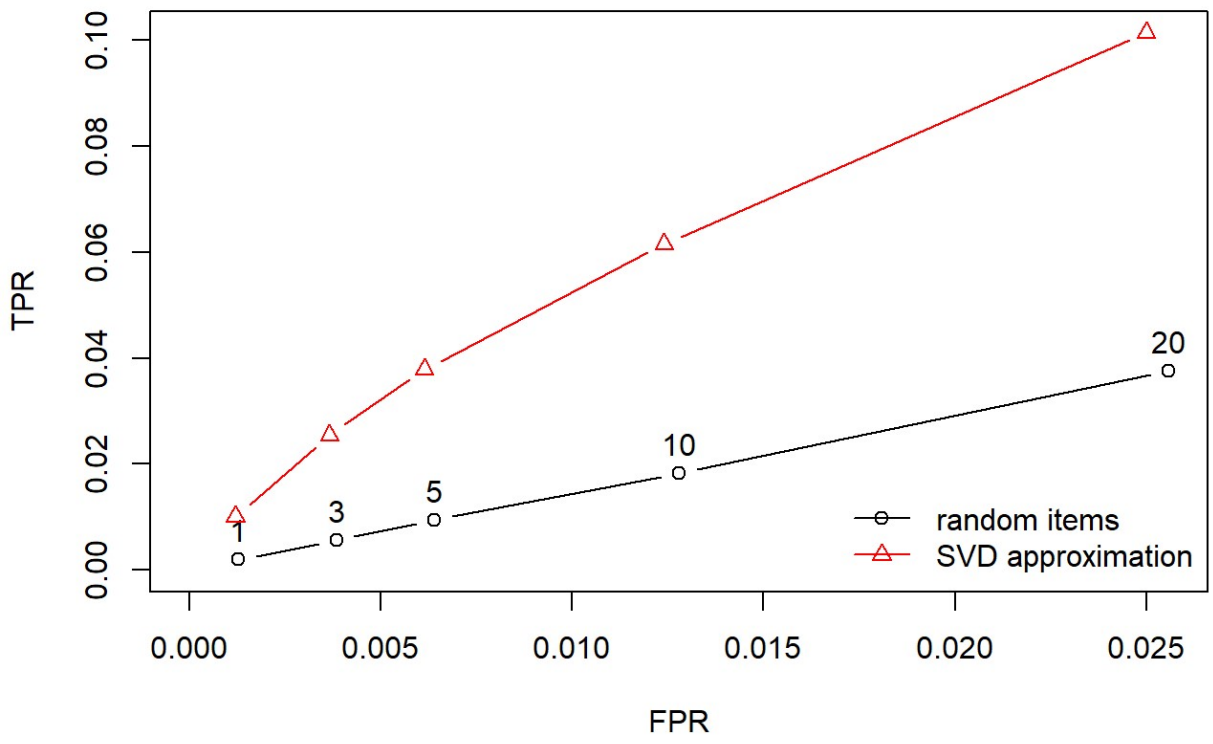
I am using the SVD and random methods in the recommenderlab package to create two models using default parameters, in order to determine if the SVD method is actually better than a random recommendation. The evaluate function can evaluate the efficiency of the algorithm for different levels of results.

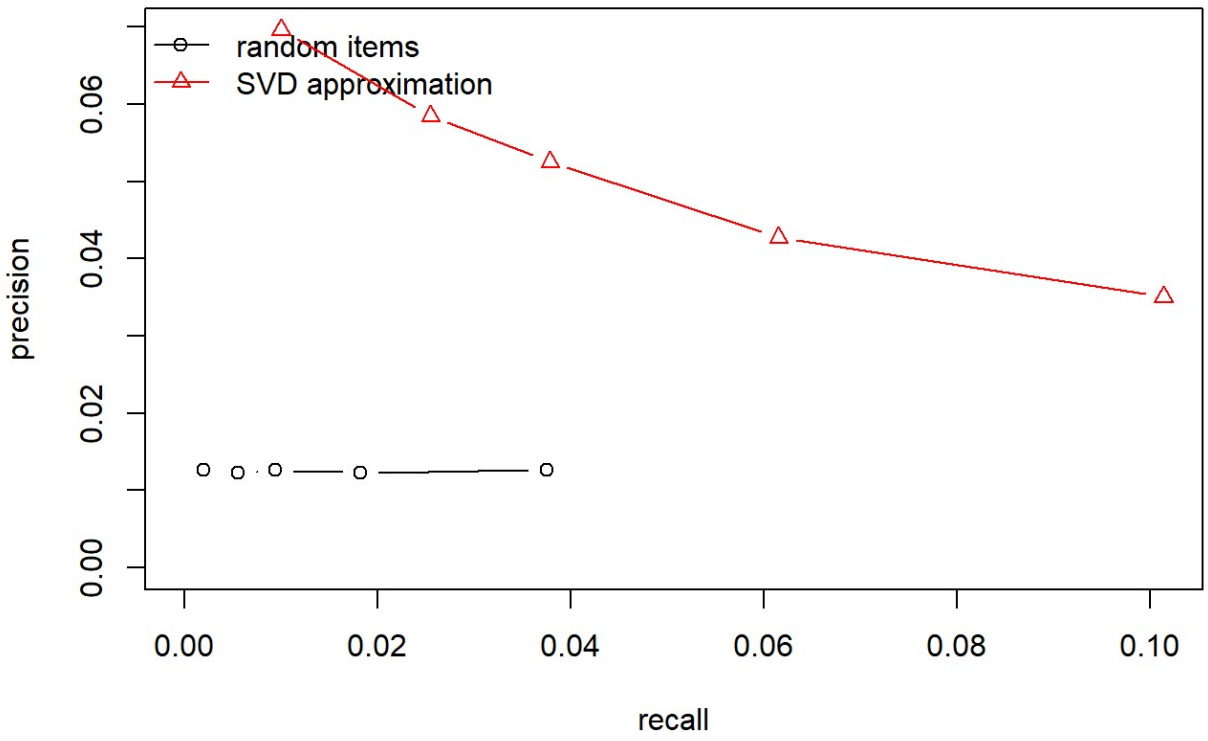
```
#train models

systems <-  list("random items" = list(name="RANDOM", param=NULL),
                 "SVD approximation" = list(name="SVD", param=NULL))

sysresults <- evaluate(evalsplit, systems, type = "topNList", n=c(1, 3, 5, 10, 20))
```

```
## RANDOM run fold/sample [model time/prediction time]
##   1  [0.02sec/0.91sec]
##   2  [0sec/0.96sec]
##   3  [0sec/0.89sec]
##   4  [0sec/0.9sec]
## SVD run fold/sample [model time/prediction time]
##   1  [0.04sec/0.99sec]
##   2  [0.03sec/0.98sec]
##   3  [0.04sec/0.73sec]
##   4  [0.03sec/0.75sec]
```

The run time for a random model was signficantly faster than that of the SVD method. If this was calculated once per day it may not be a large problem, but if it was needed to reevaluate real-time the more resource-intensive SVD method may not be scaleable.

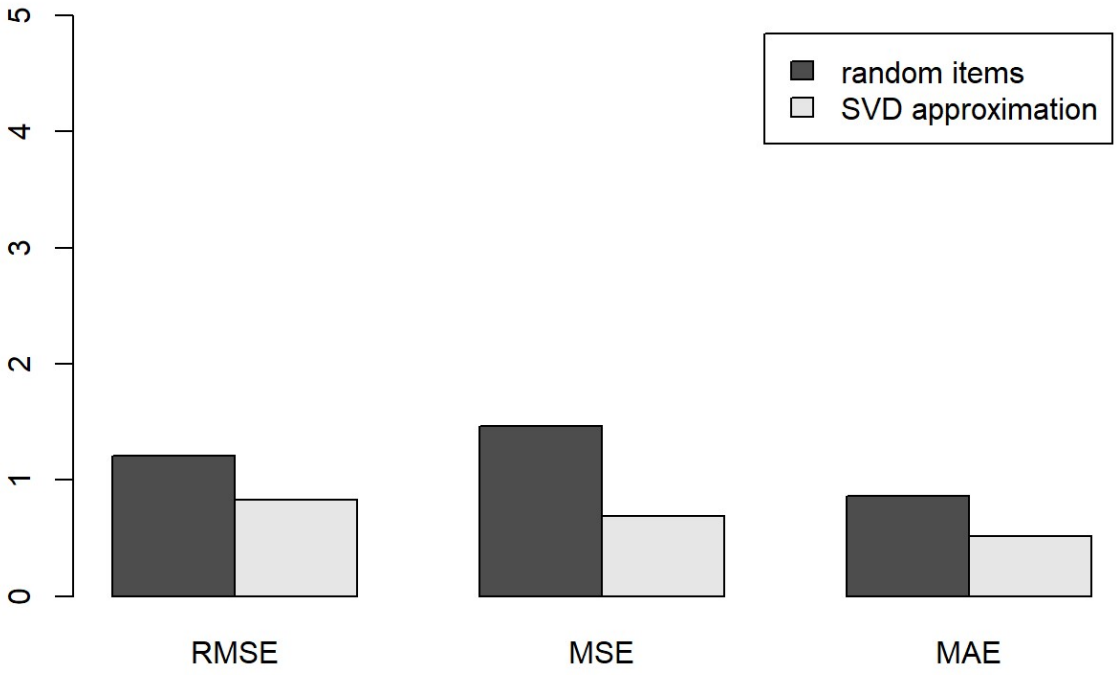
Comparing the two model's evaluation schemes using a ROC curve shows that for a lower number of recommendations the random model are fairly close in accuracy to the SVD model, but for larger number of recommendations required the less accurate the prediction of positive ratings becomes.





Further evaluating the models, we can look at how each actually predicts the rating. Comparing the errors form each model, it is interesting to note that the RMSE is not dramaticaly different betwen the random and SVD models when evaluating individual predictions.

```
## RANDOM run fold/sample [model time/prediction time]
##   1  [0sec/0.51sec]
##   2  [0sec/0.66sec]
##   3  [0sec/0.67sec]
##   4  [0sec/0.45sec]
## SVD run fold/sample [model time/prediction time]
##   1  [0.03sec/0.35sec]
##   2  [0.05sec/0.35sec]
##   3  [0.02sec/0.55sec]
##   4  [0.02sec/0.36sec]
```



This is an indication that although a random selection of items offers a decent chance of positively rated items, it does not predict the user's rating for individual items as well.

Finding that the SVD model is not hugely better than a random selection may be a result of how the data was first reduced from a very sparse matrix down to much denser data for the purpose of this evaluation. We have a densely packed matrix full of positively rated items, in which case a random selection of several may provide a fairly good recomendation and be the simplest and best model. It is certainly faster and uses less resources.

This result indicates that little information is gained by having user and item history, and would need to be examined carefully before deciding if this scneario and data is an appropriate candidate for the SVD method. The reduction of the data may have changed the true shape and this large dimension problem may need to be approached in another fashion, by partitioning the data into subsets instead of only selecting for the dense elements.