Project 4 Code

Charley Ferrari

July 11, 2016

```
library(plyr)
library(dplyr)
library(reshape2)

library(plotly)
library(Matrix)
library(knitr)

library(recommenderlab)

setwd('/Users/Charley/Downloads/cuny/IS 643 Recommender Systems/Week 4')
```

Adding Context to a Recommendation Engine

This dataset comes from a collaborative bookmark tagging platform called del.icio.ous. In this platform, users are given the ability to store bookmarks and tag them. Our main data matrix, user_taggedbookmarks, includes a collection of timestamped tags. My business case will be to recommend bookmarks to users, based on information given by their choices in bookmarks and tags. For this system, I will demonstrate the multiple dimensions I have by creating two separate recommender engines: one using tags and one using bookmarks.

I'm also going to add time as a context. I'll calculate the age of a bookmark by its first appearance in the database. I'll take the time of the recommendation into account, and weight a bookmark's recommendation by how old it is (I will also use this method to remove bookmarks that haven't been added to the system yet.)

First, I'll work on creating my two recommender systems. I'm going to design these as functions that take in a user, bookmark, and timestamp. Taking in the timestamp is important, because I'm going to use it to subset my data, only taking into account data that I would have had prior to that time.

These actions are binary, but when we're looking at only two variables, users by bookmarks or users by tags, we can get multiple counts (users can add multiple tags to a bookmark for example.) I'm going to treat this differently depending on whether we're looking at users and bookmarks or users and tags.

For users and bookmarks, increasing the number of tags and bookmarks does not increase the affinity of that user to a bookmark. The user has already expressed interest in this bookmark, and is simply adding additional classification to it. For this reason, I'm creating a binary ratings matrix that only shows 1's for when a user has tagged a bookmark, and 0 if there has been no tag.

```
user_taggedbookmarks_timestamps <- read.delim('user_taggedbookmarks-timestamps.dat',
                                              sep='\t', header=TRUE)

user_taggedbookmarks_timestamps$normTime <-
  user_taggedbookmarks_timestamps$timestamp -
  min(user_taggedbookmarks_timestamps$timestamp)

user_taggedbookmarks_timestamps$sample <-
  sample(c(0,1), nrow(user_taggedbookmarks_timestamps), replace=TRUE, prob=c(0.8,0.2))

train <- user_taggedbookmarks_timestamps %>%
```

```
  filter(sample == 0) %>% select(-sample)

test <- user_taggedbookmarks_timestamps %>%
  filter(sample == 1) %>% select(-sample)

testrow <- sample_n(test, 1)
user <- testrow[1,'userID']
bookmark <- testrow[1,'bookmarkID']
time <- testrow[1,'normTime']
window <- round(runif(1,0,1)*time)

createPredictionMatrixBookmarks <- function(time, window){

  trainUB <- train %>%
    filter(normTime < time) %>% filter(normTime > window)

  testUB <- test %>%
    filter(normTime < time) %>% filter(normTime > window)

  trainUBTFAC <- trainUB %>%
    transform(bookmarkID = factor(bookmarkID), userID = factor(userID),
              tagID = factor(tagID))

  trainBIN <- trainUBTFAC %>%
    distinct(userID, bookmarkID) %>% select(userID, bookmarkID)

  trainUBMat <- as(trainBIN, 'binaryRatingMatrix')

  testUBTFAC <- testUB %>%
    transform(bookmarkID = factor(bookmarkID), userID = factor(userID),
              tagID = factor(tagID))

  testBIN <- testUBTFAC %>%
    distinct(userID, bookmarkID) %>% select(userID, bookmarkID)

  testUBMat <- as(testBIN, 'binaryRatingMatrix')

  fsvd <- funkSVD(trainUBMat, k = 3)
  p <- recommenderlab::predict(fsvd, testUBMat)

  return(p)

}
```

By contrast, adding multiple bookmarks to a tag does increase a user's affinity for that tag. It proves they more interested in that topic matter, and are reading more links about it.

So when I'm calculating my user/bookmark and user/tag matrix, the user/bookmark matrix will include only 1's, while the user/tag matrix will include weights for each tag.

My business case will be to map this back to bookmarks, but this function will recommend a tag to a user, as input into the overall recommendation.

```
createPredictionMatrixTags <- function(time, window){

  trainUT <- train %>%
    filter(normTime < time) %>% filter(normTime > window)

  testUT <- test %>%
    filter(normTime < time) %>% filter(normTime > window)

  trainUBTFAC <- trainUT %>%
    transform(bookmarkID = factor(bookmarkID), userID = factor(userID),
              tagID = factor(tagID))

  trainUTMat <- sparseMatrix(as.integer(factor(trainUBTFAC$userID)),
                as.integer(factor(trainUBTFAC$tagID)),
                x=1)

  rownames(trainUTMat) <- levels(trainUBTFAC$userID)
  colnames(trainUTMat) <- levels(trainUBTFAC$tagID)


  trainUTMat <- as(trainUTMat, 'realRatingMatrix')

  fsvd <- funkSVD(trainUBMat, k = 3)
  p <- recommenderlab::predict(fsvd, testUBMat)

  return(p)

}
```

With this function, we can reshape our data into a bookmark by tag matrix. For example, the bookmarkID '7' Has the following "tag distribution":

```
kable(user_taggedbookmarks_timestamps %>%
  filter(bookmarkID == 7) %>%
  group_by(tagID) %>%
  summarize(count = n()))
```

tagID	count
1	2
6	1
7	1
13	1
18	1
19	1
20	1
21	1

For additional information on a bookmark's probable ranking, I would be able to use this information. However, since these are counts, making proper use of them would require normalizing the data in some way, which would probably be even more computationally expensive than running the recommender for each tag in the tag distribution. This might need to be pre-processed and ran sparingly if it's ever going to be useful.

Lastly, throughout my functions, time is used as a context. By adding the timestamp, I create the train and test data based on whether they occurred before that specific time. I also added a "window" to throw out old ratings as the recommender system runs over time.

This is meant to account for tastes changing over time. Older ratings may be stale, and users can change over time. Ideally, I would like to add a "decay" factor for ratings, allowing ratings to have lower weightings if they occurred a long time ago.