MSiA 421, Data Mining
Content-Based Recommenders
Due: Monday, March 2, 5pm

In this assignment, you will hand-create and use some content-based profiles. You'll go through a set of variations to see how certain features of the computation can introduce (or reduce) biases.

1. The Data Set. First, download the `cbf.xlsx` dataset. It contains a table of content attributes for 20 documents across 10 attributes (cells A1:K21). It also lists two users' evaluations of five documents each (cells O1:P21). For purposes of this assignment, we're treating content attributes as boolean (either an article is about a topic or it isn't) and we're treating evaluations as positive (liked it), negative (disliked it), or unknown (never saw it).

   Build and use a very basic profile. First, you will build a very simple profile of user preferences for attributes. In this profile, you'll count the total the number of positive and negative evaluations associated with each attribute, and create a profile with the total score for each attribute for each user. For example, user 1's score for "Family" will get a +1 from doc1 (positive evaluation) and a −1 from doc 19 (negative evaluation) for a total profile value of 0 (neutral). In contrast, user 2's score for Europe will be +3 (+1 each for doc2, doc4, and doc17). You can compute the profiles and place them in the "User Profiles" section of the spreadsheet (cells `A26:K28`). Hint: use the `SUMPRODUCT` function in Excel.

   Now compute the predicted score (cells `R1:S21`) for each user for each document (a simple dot product).

   (a) Which document does the simple profile predict user 1 will like best? *Answer: 16*

   (b) What score does that prediction get? *Answer: 6*

   (c) How many documents does the model predict user 2 will dislike (prediction score that is negative)? *Answer: 4*

   Notice that this model is consistent with the users' ratings—it predicts liking for all the positive documents and disliking for all the negative ones.

2. Next, let's treat all articles as having unit weight. You may have noticed that in our computation an article that had many attributes checked could have more influence on the overall profile than one that had only a few. Doc1 and doc19 each have five attributes, while doc6, doc7, and doc18 only have 2 attributes each. Add counts of the number of attributes in cells `M1:M21` (e.g., use the `SUM` or `COUNT` function).

   We want to explore whether our simple model may be counting these attribute-heavy documents too much. For example, we might conclude that liking doc6 says more about liking baseball (since it is one of only two attributes for the article along with

Europe) than liking doc1 says (since doc1 is also about politics, Asia, soccer, and family). To try this out, make a copy of the attributes matrix in cells A32:K52. Then you will normalize each row to be a unit length vector. We can do this in two steps:

Divide each value in the original data by the square root of that number of items. If you do this right, doc1's values will all change from 1 to ≈ 0.447214. Documents with 4 attributes will change to 0.5 (since $4 \times .5^2 = 1$), and so forth. Remember, don't have the `SUM` or `COUNT` depend on the copy of the cells you're changing or you'll get a circular dependency. Have your new block depend on values on your old one. Once you have the new values, compute your second set of user profiles (cells A55:K56) and new predictions (cells R32:S52). If you did this right, you'll see a prediction of ≈ 1.0090 for user1/doc1 in cell R33. Don't worry about the scale of the numbers (they'll all be smaller, in absolute value terms), but look at the order of them.

This time we'll start with user2. With our simple profile, doc7 and doc19 both had similar "like" predictions (+2 each). Now they don't. Let's see what values are predicted for the second model. Here are two values to check your work: doc7, (S39) 0.7444 (± 0.01); doc 19, (S51) 0.4834 (± 0.01).

The difference here can be seen by looking at the profile attribute values. Doc7 is 50% about one of user2's favorite topics (security) which is now more heavily emphasized).

Now let's look at user 1. While user 1's first-place document is the same in both models, that isn't true for other places. In our simple model, the second/third place recommendation was a tie between doc1 and doc12. Neither of those is in second place with this new model.

(a) Which document is now in second with this new model? *Answer: 6*

(b) What prediction score does it have? *Answer: 1.370922666*

3. In this problem, you will implement user-user collaborative filtering using a spreadsheet. Or you could do this in R "by hand" using only the basic functions. First, you will implement user-user collaborative filtering without normalization.

   (a) Start by downloading the starting spreadsheet. This is a 25 user $\times$ 100 movie matrix of ratings selected from the class data set. The spreadsheet has three sheets in it (this is not supposed to be an exercise in spreadsheet tricks; as a result, I've already given you a significant start). The first sheet is a ratings matrix with movies as rows and users as columns. The second sheet is a ratings matrix with movies as columns and users as rows. The third sheet is the start of your correlations matrix.

   (b) Open the sample matrix in your favorite spreadsheet program. Note that the matrix contains a significant number of missing values—do not replace these with zeroes, they are correctly missing.

(c) Complete the user-by-user correlations matrix. To check your math, note that the correlation between users 1648 and 5136 is 0.40298, and the correlation between users 918 and 2824 is -0.31706. All correlations should be between -1 and 1, and the diagonal should be all 1's (since they are self-correlations). R Hint: `cormat = cor(R, use="pairwise")`

(d) Identify the top 5 neighbors (the users with the 5 largest, positive correlations) for users 3867 and 89. For example, if the target user were number 3712, the closest neighbors are 2824 (corr: 0.46291), 3867 (corr: 0.400275), 5062 (corr: 0.247693), 442 (corr: 0.22713), and 3853 (corr: 0.19366). Don't forget to exclude the target user (corr: 1.0) from your possible selections. *Answer: 3867: 2492, 3853, 2486, 3712, 2288; 89: 4809, 5136, 860, 5062, 3525*

(e) Create a new worksheet in your spreadsheet, and use it to compute the predictions for each movie for users 3867 and 89 by taking the correlation-weighted average of the ratings of the top-five neighbors (for each target user) for each movie. The formal formula for correlation-weighted average is

$$\frac{\sum_{n=1}^{5} r_n w_n}{\sum_{n=1}^{5} w_n}.$$

Remember, you will need to make sure that your weight for each contributed rating is the user-user correlation when that neighbor has rated the movie, but 0 when the neighbor has not rated the movie).

(f) Submit 12 values for this assignment as indicated below. You will be submitting the top three movie IDs and the predicted ratings (to three decimal places) for each user. In a real recommender system, we'd be excluding movies the user has already rated, but do not do this here. Indeed, you should look to see what the user's rating (if any) is for the top-recommended movies. For example, if the user ID was 3712, the correct submission would be:

- Top Movie: 641 Prediction: 5.000
- 2nd Movie: 603 Prediction: 4.856
- 3rd Movie: 105 Prediction: 4.739
- And if the user ID were 3525, there would be a three-way tie among:
- Movies: 238, 194, and 38 (all with a prediction of 5.000)

A few tips we hope you'll find helpful:

- Your calculations will be easier if you use some of the built-in functions that spreadsheets have to help. These include `SUMPRODUCT` (which computes dot-products) and `SUMIF` (which can be used to compute the total weight). It is also helpful to understand two special paste functions: paste values (which copies cell values without copying the underlying formulas) and paste transpose (which can help reorient your data).

- Your intermediate goal should be to have a sheet for each target user with a user-by-rating table and a set of user weights (for the top-five neighbors) to allow you to compute all 100 predictions for the target user.
- Once you have all the predictions, you can always copy them (copy the values) and sort them to find the top three results.
- Don't worry about values that result in division by zero—those are cases where no neighbors rated the movie.
- It will take 95% of the work to get a single instance of this correct. Use the test cases to make sure you have your computations correct. Then compute the cases you'll submit.

Answer: Here is my answer:

```
---------------------------------------------------------------
neighbors for  X3867 :X2492 X3853 X2486 X3712 X2288
    X2492     X3853     X2486     X3712     X2288
0.4766833 0.4641101 0.4389916 0.4002745 0.3798563
Without normalization
           1891: Star Wars V (1980)        4.760
155: The Dark Knight (2008)                4.551
122: The Lord of the Rings 3 (2003)        4.508

With normalization
   1891: Star Wars V (1980)          5.246
155: The Dark Knight (2008)          4.857
77: Memento (2000)                   4.778
---------------------------------------------------------------
neighbors for  X89 :X4809 X5136 X860 X5062 X3525
    X4809     X5136      X860     X5062     X3525
0.6685160 0.5624487 0.5390659 0.5259904 0.4754949
Without normalization
       238: The Godfather (1972) 4.894
278: Shawshank Redemption (1994) 4.882
               807: Seven (1995) 4.774
With normalization
       238: The Godfather (1972) 5.322
278: Shawshank Redemption (1994) 5.261
               275: Fargo (1996) 5.241
---------------------------------------------------------------
Here is my R code
R = read.csv("/Users/ecm/teach/RecSys/MinnRecSys/asgn/hw3/uucf.csv")
row.names(R) = R$X
R$X=NULL
head(R)
```

```
dim(R)
users = names(R)
cormat = cor(R, use="pairwise")
cormat

# r = rating vector of only the nearest neighbors
# w = weight vector (corrs) of only NNs
myscore = function(r, w){
  ok = !is.na(r)
  sum(r[ok]*w[ok])/sum(w[ok])
}
# same as above, but rbar is user means for only NNs,
# and rbar_u is user mean for active user
myscoreN = function(r, w, rbar, rbar_u){
  ok = !is.na(r)
  rbar_u + sum((r[ok]-rbar[ok])*w[ok])/sum(w[ok])
}
rbar = apply(R, 2, mean, na.rm=T)

for(u in c(7, 9, 5, 14)){
    cat("neighbors for ", users[u], ":")
  ord = order(-cormat[,u])
  cat(users[ord[2:6]], "\n")
  print(cormat[ord[2:6], u])
  cat ("Without normalization\n")
  Rhat = apply(R[, ord[2:6]], 1, myscore, w=cormat[ord[2:6], u])
  ord2 = order(-Rhat)
  print(round(Rhat[ord2[1:3]], 3))
  cat ("With normalization\n")
  Rhat = apply(R[, ord[2:6]], 1, myscoreN, w=cormat[ord[2:6], u],
    rbar=rbar[ord[2:6]], rbar_u=rbar[u])
  ord2 = order(-Rhat)
  print(round(Rhat[ord2[1:3]], 3))
  cat("--------------------\n\n")
}
```

4. Next, you will repeat the computation but this time you will normalize the scores.

   (a) Repeat step 5 from part 1. This time, however, use the normalization formula:

$$\bar{r}_u = \frac{\sum_{n=1}^{5}(r_n - \bar{r}_n)w_n}{\sum_{n=1}^{5} w_n}$$

   (b) Submit 12 values for this assignment as indicated below. You will be submitting the top three movie IDs and the predicted ratings (to three decimal places) for

users 3867 and 89. In a real recommender system, we'd be excluding movies the user has already rated, but do not do this here. Remember, you do not need to re-compute the correlations, just use the existing correlations but normalize the ratings being averaged by subtracting each neighbor's mean rating from each of their ratings (and add the target user's mean back into the total).

For example, if the user ID was 3712, the correct submission would be:

- Top Movie: 641 Prediction: 5.900
- 2nd Movie: 603 Prediction: 5.546
- 3rd Movie: 105 Prediction: 5.501

Note that it is possible to have movie predictions outside the five-star scale; in this case this is because 3712 rates movies very high to begin with, and his/her neighbors have wider ranges (and think these movies are very high in those ranges).

Also, these top movie predictions are interesting in other ways. The top movie was only rated by one neighbor, but was so highly rated (1.4 above mean) that it stands out.

And if the user ID were 3525, the top three would be:

- Top Movie: 238 Prediction: 4.760
- 2nd Movie: 424 Prediction: 4.663
- 3rd Movie: 134 Prediction: 4.585

Note how only one of these three is the same as from the top three without normalization! And notice that the (perhaps surprisingly) strong prediction for 134 is again the result of having just a single neighbor's rating (and that being a neighbor who really liked the movie).

Can you figure out how to do this in R using recommenderlab and Surprise in Python.