PROP, algorithms part, Guillem Godoy

This document is concerned with the part of algorithms of the project of the PROP subject, first semester, course 2021-2021. It does not contain additional information with respect to what has already been published in the website of the subject; but it is just an attempt to interpret such information and make it more accessible to the students.

Disclaimer: this text is exclusively addressed to the students of group 21, and its contents must be taken with a grain of salt, as it's not been reviewed by the whole set of professors teaching this matter. It is just aimed at being of help in order to undertake this part of the project with a better insight.

1 Files/Data

We have the following files/data:

- `items.csv` represents the matrix $\mathbf{Items}[j][a]$ for itemid j and attribute a. The file contains lines $j, \mathbf{Items}[j][a1], \ldots, \mathbf{Items}[j][an]$ representing all the values.
- `ratings.db.csv` represents the matrix $\mathbf{Ratings}[i][j]$ for userid i and itemid j. This matrix is sparse and the file contains lines $i, j, \mathbf{Ratings}[i][j]$ representing the defined values.
- `ratings.test.known.csv` represents the matrix $\mathbf{Known}[i][j]$ for userid i and itemid j in a similar way to **Ratings**. The involved userid's are disjoint with those in **Ratings**.
- `ratings.test.unknown.csv` represents the matrix $\mathbf{Unknown}[i][j]$ for userid i and itemid j in a similar way to **Ratings**. The involved userid's are exactly the same as those in **Known**.

2 General behaviour, input and output of the algorithms

Any of the algorithms should work in two phases. First, they receive **Items** and **Ratings** as data for preprocessing. Second, they are asked queries of the form:

$$\mathbf{userid},\ (\mathbf{itemid}_1, \mathbf{rating}_1), \ldots, (\mathbf{itemid}_d, \mathbf{rating}_d),\ \mathbf{itemid}'_1, \ldots, \mathbf{itemid}'_q,\ Q.$$

where **userid** is not in the domain of **Ratings** and $Q \leq q$. This input represents a new user that has rated items $\mathbf{itemid}_1, \ldots, \mathbf{itemid}_d$ as described, and for whom we want to predict the Q preferred items of this user among $\mathbf{itemid}'_1, \ldots, \mathbf{itemid}'_q$, and their order of preference.

For each query, the algorithm should produce a sorting/permutation of a subset of Q elements of $\mathbf{itemid}'_1, \ldots, \mathbf{itemid}'_q$ representing the recommendation order from best to worst of those items for that user.

The queries are built by making use of **Known** and **Unknown**. For each userid i in the domain of those matrices we build the query

$$i,\ (j_1, \mathbf{Known}[i][j_1]), \ldots, (j_d, \mathbf{Known}[i][j_d]),\ j'_1, \ldots, j'_q, Q.$$

where $j_1, \ldots, j_d$ are all the items j holding that $\mathbf{Known}[i][j]$ is defined, and $j'_1, \ldots, j'_q$ are all items j holding that $\mathbf{Unknown}[i][j]$ is defined, and Q is randomly chosen among $\{1, \ldots, q\}$.

3 Assessment of the results of the algorithms

In order to determine how good the resulting permutation from the algorithm is, we need to define the concept of Discounted Cumulative Gain (**DCG**) for a given user i, and a given permutation $P = j_1, \ldots, j_q$ holding that $\mathbf{Unknown}[i][j_1], \ldots, \mathbf{Unknown}[i][j_q]$ are defined values for user i in **Unknown**, as:

$$\mathbf{DCG}_{i,P} = \sum_{d=1}^{q} \frac{2^{\mathbf{Unknown}[i][j_d]} - 1}{\log_2(d+1)}$$

and that's the value that we use to measure the accuracy of the resulting permutation from the algorithm.

Optionally, in order to get a more canonical number (between 0 and 1), we can define the ideal **DCG** for user i and q elements, called $\mathbf{IDCG}_{i,q}$. With that purpose, the define the best permutation $P_{i,q,\mathbf{best}}$ for i as that obtained by sorting the items j for which $\mathbf{Unknown}[i][j]$ is defined in decreasing order of their values $\mathbf{Unknown}[i][j]$ and choosing the first q. Then:

$$\mathbf{IDCG}_i = \mathbf{DCG}_{i,q,P_{i,\mathbf{best}}}$$

Finally, the normalised **DCG** is defined as:

$$\mathbf{NDCG}_{i,P} = \frac{\mathbf{DCG}_{i,P}}{\mathbf{IDCG}_{i,|P|}}$$

4 Case of Kmeans+Slope1

In this case, the preprocessing consists of running kmeans with the aim to compute a partition $P_1, \ldots, P_k$ of the set of users occurring in **Ratings** and the corresponding centroids $c_1, \ldots, c_k$.

For a given query:

$$i,\ (j_1, \mathbf{Known}[i][j_1]), \ldots, (j_d, \mathbf{Known}[i][j_d]),\ j'_1, \ldots, j'_q, Q.$$

we first find the closest centroid c_s to user i. Then, the corresponding part P_s is used to apply Slope1 q times in order to compute the predicted ratings $\mathbf{Predicted}[i][j'_1], \ldots, \mathbf{Predicted}[i][j'_q]$. Finally, we sort $j'_1, \ldots, j'_q$ from biggest to smallest predicted value and choose the first $|Q|$ elements.

4.1 Distances

The above processes need to compute distances between users based on matrix **Ratings** and **Known**. To make it simpler, let's assume that **Known** has been added as part of **Ratings** and consider two users u, v. Note that $\mathbf{Ratings}[u], \mathbf{Ratings}[v]$ are the corresponding rows for those users. A possible definition that we could use is:

$$\mathbf{Distance}(u,v) = \sqrt{\sum_{j:\mathbf{Def}(\mathbf{Ratings}[u][j]) \wedge \mathbf{Def}(\mathbf{Ratings}[v][j])} (\mathbf{Ratings}[u][j] - \mathbf{Ratings}[v][j])^2}$$

4.2 Kmeans

The partition and centroids are computed by considering the following process: first, initialise $c_1, \ldots, c_k$ as the rows of k users chosen randomly. If some of their components are undefined, then they are initialised randomly between the minimum and maximum possible rating. Second, run the following process a certain number of times:

1. Compute the partition $P_1, \ldots, P_k$ of the set of users, where P_i contains the users that are closest to c_i than to any other centroid.
2. Recompute $c_1, \ldots, c_k$ by setting c_i to the average of the points of P_i. Only defined values are considered for each component. If there are no defined values in that part for a specific component, then a random value is chosen for it.

The final $P_1, \ldots, P_k$ and $c_1, \ldots, c_k$ are the result of Kmeans.

4.3 The 3-definitions condition

In order to describe slope1 in an easy way, it will be good to define first the 3-definitions condition for given i, j, I, J:

$$\mathbf{3DEF}(i,j,I,J) = \mathbf{Def}(\mathbf{Ratings}[i][j]) \wedge \mathbf{Def}(\mathbf{Ratings}[i][J]) \wedge \mathbf{Def}(\mathbf{Ratings}[I][j])$$

4.4 Weighted Slope1

If we use Weighted Slope1 instead of Slope1, the formulation of the algorithm for computing $\mathbf{Predicted}[I][J]$ for given I, J (assume that $\mathbf{Known}[I]$ has been added as part of **Ratings**) is quite easy: we initialise two values **SUM** and **NUMBER** to 0 and consider all i, j holding $\mathbf{3DEF}(i, j, I, J)$ and do:

$\mathbf{SUM} += \mathbf{Ratings}[I][j] + (\mathbf{Ratings}[i][J] - \mathbf{Ratings}[i][j])$
$\mathbf{NUMBER}++$

Finally, $\mathbf{Predicted}[I][J] = \frac{\mathbf{SUM}}{\mathbf{NUMBER}}$

4.5 Slope1

Alternatively, if we use Slope1, $\mathbf{Predicted}[I][J]$ requires first computing $\mathbf{Predicted}_{IJ}[j]$ for each j such that there exists i holding $\mathbf{3DEF}(i, j, I, J)$ as follows: we initialise two values **SUM** and **NUMBER** to 0 and consider all i holding $\mathbf{3DEF}(i, j, I, J)$ and do:

$\mathbf{SUM} += \mathbf{Ratings}[I][j] + (\mathbf{Ratings}[i][J] - \mathbf{Ratings}[i][j])$
$\mathbf{NUMBER}++$

Finally, $\mathbf{Predicted}_{IJ}[j] = \frac{\mathbf{SUM}}{\mathbf{NUMBER}}$

Once all those $\mathbf{Predicted}_{IJ}[j]$ have been computed, $\mathbf{Predicted}[I][J]$ is defined to be the average of them all.

5 Case of KNN

In this case, the preprocessing can be anything that makes it easy to compute the k-nearest neightbours to a given new user afterwards.

For a given query:

$$i,\ (j_1, \mathbf{Known}[i][j_1]), \ldots, (j_d, \mathbf{Known}[i][j_d]),\ j'_1, \ldots, j'_q,\ Q.$$

First of all, we will remove from those $(j_1, \mathbf{Known}[i][j_1]), \ldots, (j_d, \mathbf{Known}[i][j_d])$ the pairs for which the corresponding **Known** value is too low (up to a certain threshold), hence $d' \leq d$ elements remain. Next, for each remaining $(j, \mathbf{Known}[i][j])$, we compute the k closest elements to j among $j'_1, \ldots, j'_q$ and we produce pair $(S, \mathbf{Known}[i][j])$, where S is a subset of k elements from $j'_1, \ldots, j'_q$.

So far, we have obtained a collection of pairs $(S_1, value1), \ldots, (S_{d'}, value_{d'})$. Finally, we sort $j'_1, \ldots, j'_q$ from best to worst by comparing them according to how many times they appear in the subsets of those pairs and appropriately considering the corresponding values.

The best way to compute distances between items, and the way the j' should be sorted is open to discussion.