מטלת בית מס' 3

הנחיות:

- הגשה:
- יש להגיש את המטלה עד לתאריך 20.6.18.
- ההגשה תתבצע ע"י קובץ zip המכיל את הקוד שנכתב בהתאם לדרישות. פורמט קובץ ה-zip יהיה ID01_ID02.zip. שם קובץ ההרצה יהיה מותאם לכל חלק בעבודה part1.py וכן part2.py.
- ההגשה הינה **בזוגות**.
- חבר קבוצה אחד בלבד יעלה את הפתרון לאתר.
- בעיות אישיות בנוגע למועד ההגשה יש להפנות לבודק התרגילים הקורס טרם מועד ההגשה.
- **כל חריגה מנהלים אלו, ללא אישור בכתב מצוות הקורס, מהווה עילה לפסילת המטלה או להפחתת נקודות.**
- **אין להעתיק פתרונות ואין לשתף קוד בין סטודנטים. אין להעתיק קוד מוכן באינטרנט!**
- **לפתרון המטלה יש להשתמש בגרסאת פייטון 2.7 בלבד.**
- להבהרות, הכוונות או כל עזרה אחרת, ניתן לשאול שאלות בפורום המתאים למטלה זו באתר הקורס.
- בדיקת המטלה תתייחס בין השאר לפרמטרים הבאים: נכונות הקוד, יעילות הקוד וזמני ריצה. יש לבדוק מקרי קצה.

This home assignment is divided to two parts:

- The first part includes implementing user profiles and item profiles based on rating data (movilens).
- The second part includes implementing algorithm for co-clustering according to the profiles obtained in first part.

You should submit one zip or rar file.

The archived file should contain 2 files:

- The source code for "part 1".
- The source code for "part 2".

The archive file name should be of form id1_id2 (zip or rar), where "id" is your ID number.

You can use any API available online, but **you have to implement each algorithm specified here** by **yourself**. Grades will be given by **efficiency of the solution (run time performance).**

<u>Part 1 – 15%</u>:

In this part you will only use "ratings.csv" file from lab1.

In this part you will write a program which takes as an input a file ("ratings.csv") of Collaborative Filtering (CF) rating records <*userId, itemId, rating, timestamp*> and transfer the records into "user profiles" and "item profiles" (each output in a separate csv file).

In this task you should output two csv files, each represent user profiles and item profiles respectively:

- User profiles are projections of the baseline ratings from the user point of view: each line contains the userId and his relevant items/ratings. Each record is structured as <userId, array_of_itemIds, array _of_ratings>. The two lists are corresponding to each other.

  In this part it is mandatory to use Numpy library.
- Item profiles records are symmetric and include < itemId, array_of_userIds, array _of_ratings>.

  In this part it is mandatory to use Numpy library.

For example, the following baseline ratings:

1	1	2	10001
1	2	5	10002
2	2	4	10003

Will generate "user profiles.csv" file as:

1	[1,2]	[2,5]
2	[2]	[4]

And "items profiles.csv" file as:

1	[1]	[2]
2	[1,2]	[5,4]

Notes:

1. The resulted lists don't have to be sorted, but have to be implemented with numpy.
2. Implement your main method to take three parameters:

   Python Part1.py ExtractProfiles [the rating input file] [user profile csv output directory] [item profile csv output directory].
3. Run and test your program with the attached ratings.csv file from lab1.
4. Extracted files of this part have been uploaded to the website.

<u>Part 2</u> – 85%:

In this part you will implement an updated web service that recommends n=10 movies to a specific user.

requests.post("http://127.0.0.1:5000", data={'n': 10, 'userid': 1234})

The web service will be based on an offline evaluation of the algorithm called "Co-Clustering". The algorithm takes as input a dataset of ratings (userId, itemId, score), and in iterative process clusters (groups) similar users and items together. User cluster ids are notated with "K" prefix, and item cluster ids are notated with "L" prefix.

The algorithm generates a code-book: a matrix specifying how users from a certain user-cluster will rate item in a certain item-cluster, see an example in the Fig.1:

U:

UserId	1	2	3	4	5
UserClusterId (K)	0	3	1	2	3

V:

ItemId	1	2	3	4	5
ItemClusterId (L)	0	2	1	0	3

B:

	L0	L1	L2	L3
K0	2	2	1	4
K1	4	5	5	5
K2	3	3	3	5
K3	2	4	3	5

Fig.1: Code Book example.

- The "U" vector specifies for each user - which user-cluster he belongs to (e.g.: "user 1" belongs to "cluster K0").
- The "V" vector is symmetric from the items point of view (e.g.: item 5 belongs to "cluster L3").
- We assume that each user/item belongs to exactly one cluster ("hard clustering").
- "B" is code-book matrix specifying the rating patterns between the clusters, e.g.: users from type K0, will rate items from type L3 with rating score of "4" (see the right most square in the first row of B).
- Notice that these data structures will be used to give a prediction score:
  given a user, you will iterate over all items that the user didn't rate and for each of the item ids –use cluster indices (from U and V), and retrieve the rating from B. e.g.: the prediction of "user 1" on "item 5" will be "4". Formally it is notated by $B[U_{uid}][V_{iid}]$.
  Finally, by applying the web service you will retrieve the top n=10 items with the highest predictions.

Algorithm Overview:

Given a ratings dataset, this algorithm generates a codebook "B" - a KxL numpy matrix specifying the common rating for each user-item group. In addition, by-products of this algorithm are users and items clusters maps: U is a numpy vector specifying for each user his associated user-cluster index (ranges from 0 to K-1); and V - a symmetric numpy vector, mapping each item to item-cluster index (0 to L-1).The CodeBook is generated via the general co-clustering approach suggested in [1] by solving the optimization problem presented in equation "1", which aims to minimize the error between the actual rating in the training dataset (ST) and the relevant codebook pattern.

$$\min_{U,V,B} \sum_{ST} (ST_{uid,iid} - B\,[U_{uid}][V_{iid}])^2 \quad (1)$$

Algorithm Description:

Follow the attached pseudo code. In lines 1-5 we set a random cluster index for each user and item in ST, and calculate the initial code book: each element B_{ij} in the matrix is set to the average rating of all the relevant samples in ST as presented in equation "2" (i.e.: the average of all user-item pairs, where the user-cluster index is set to i, and item-cluster index is set to j).

$$B_{i,j} = \frac{\sum_{ST/\,U_{uid}=i \wedge V_{iid}=j} ST_{uid,iid}}{|\,ST/\,U_{uid}=i \wedge V_{iid}=j\,|} \quad (2)$$

- In cases when $|\,ST/\,U_{uid}=i \wedge V_{iid}=j\,|$=0, Bi,j is set to the <u>average rating</u> of the system.

Next, we start an iterative process of co-clustering and pattern discovering (lines 6-15). Each of the iterations consists of three parts:

- Users Cluster Discovery: according to the current V and B values we find the optimal cluster-index for each user (lines 8-9), minimizing the total error between the user's actual ratings and the predicted ones using the current model. After discovering the new clusters for all users, we update the codebook (11).
- Items Cluster Discovery – symmetric to previous phase (lines 11-13), but from the items' perspective.
- Evaluation - we evaluate accuracy the current model ($U^{(t)}$,$V^{(t)}$ and $B^{(t)}$), by calculating the Root Mean Square Error (RMSE) measure on a validation set - SV (line 14).

The termination condition is either:

- After predefined T iterations.
- After iteration with no significant improvement the RMSE compared to the previous iteration (improvement which is less than a threshold value ε, or no improvement at all).

Lastly, the algorithm returns the generated model from the iteration with minimum RMSE (line 16).

Pseudo Code:

CodeBook Construction

Input:

- *ST –Source domain training rating dataset<u, i ,r>. User ids range from 1 to p, and Item ids range from 0 to q.*
- *SV - Source domain training validation dataset<u, i ,r> (similar structure to ST).*
- *K - Number of possible user clusters.*
- *L - Number of possible item clusters.*

Output:

- *B – Source domain CodeBook a KxL matrix specifying for each user-cluster-index and item-cluster-index pair the rating pattern according the values in ST).*
- *U – Source domain user vector, specifying for each user the relevant cluster index according to the given CodeBook.*
- *V – Source domain item vector, specifying for each item the relevant cluster index according to the given CodeBook.*

Method:

1. For iid←1 .. q do
2. Set randomly $V^{(0)}_{iid}$ from {0, .. *L-1* }
3. For uid←1 ..p do
4. Set randomly $U^{(0)}_{uid}$ from {0, .. *K-1* }
5. Calculate current CodeBook – $B^{(0)}$, using update rule in equation "2".
6. Set t =1
7. While (*!Termination_Condition*) do
8. For uid←1 ..p do
9. $$U^{(t)}{}_{uid} = \arg\min_{j,\, j\in\{0\ldots K-1\}} \sum_{iid\in ST_{uid}} (ST_{uid,iid} - B^{(t-1)}[j][V^{(t-1)}_{iid}])^2$$
10. Calculate current CodeBook – $B^{(t)}$, using update rule from equation "2".
11. For iid←0 .. q-1 do
12. $$V^{(t)}{}_{uid} = \arg\min_{j,\, j\in\{0\ldots L-1\}} \sum_{uid\in ST_{iid}} (ST_{uid,iid} - B^{(t)}[U^{(t)}_{uid}][j])^2$$
13. Update the current CodeBook – $B^{(t)}$, using update rule from equation "2".
14. Evaluate current model (*calculate RMSE*) on *SV* data set
15. t = t+1.
16. Return $U^{(t)}$, $V^{(t)}$, $B^{(t)}$ with minumum RMSE on *SV*.

Fig 2 illustrates how the algorithm updates the cluster of a certain user (line 9), given the present model values:

U:

UserId	1	2	3	4	5
UserClusterId (K)	0	3	1	2	3

V:

ItemId	1	2	3	4	5
ItemClusterId (L)	0	2	1	0	3

B:

	L0	L1	L2	L3
K0	2	1	1	4
K1	4	5	5	5
K2	3	3	3	5
K3	2	4	3	5

ST:

UserId	ItemId	Rating
1	1	3
1	3	5

Fig.2: Code Book Update example.

- The goal is to find the current optimal cluster id of "user 1", given the current V and B.
- The profile (ST) of "user 1" contains two items (1, 3) with ratings of (3,5) respectively.
- Given the profile and the current V and B, we iterate all possible user-clusters options (K0, K1, K2 and K3), and select the one with minimum squared error between the actual rating and the predicted rating:
    - K0 – if we assign the user to K0, the predicted ratings for items 1 and 3 will be 2 and 1 receptively, so the total error is: $(3-2)^2 + (5-1)^2 = 17$.
    - K1 –> total error: $(3-4)^2 + (5-5)^2 = 1$.
    - K2 –> total error: $(3-3)^2 + (5-3)^2 = 4$.
    - K3 – >total error: $(3-2)^2 + (5-4)^2 = 2$.
- The minimum error is with K1 - this is the selected cluster for "user 1".

<u>Some Design Notes</u>:

- Most of the computations are performed during the iterative processes - design your algorithm with **minimum** time and maximum level of parallelism/distribution.
- Design your algorithm to be configurable, default values are: K=L=20; T=10; ε=0.01.
- Implement your main method to take 7 parameters:
  Python Part2.py ExtractCB [the rating input file] [K size] [T size] [ε size] [U output directory as csv file] [V output directory as csv file] [B output directory as csv file]. If T or ε are null- default values should be defined as explained.
- Implement a web service which uses the extracted Codebook in order to retrieve the top-n items for this user with the highest predictions:
  requests.post("http://127.0.0.1:5000", data={'n': 10, 'userid': 1234})

- In order to compute the RMSE, the size of SV (validation set) is 20% of all rating data. (We should split the input data to 20% (ST) and 80% (SV))

Additional Reading:

1. Papadimitriou, Spiros, and Jimeng Sun. "Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining." *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 2008.
2. Li, Bin, Qiang Yang, and Xiangyang Xue. "Can movies and books collaborate? cross-domain collaborative filtering for sparsity reduction." *Proceedings of the 21st international jont conference on Artifical intelligence*. Morgan Kaufmann Publishers Inc., 2009.
    a. Note: The code book construction parts.