Homework 3

CSCI 4907/6444: Big Data and Analytics

Fall 2023

The deadline for homework assignment 3 is **11/14/2023 11:59 pm**. The total number of points for this assignment is 10. Make sure that you upload a *single* zip file for all of the questions in the assignment on blackboard. Create two folders `q1` and `q2` and put all the related content (code, output files, writeups, etc.) in the corresponding folder. Do not include the input data in your submissions. Name you folder `lastname_firstname_hw3` and compress it to create a zip file. Upload your zip file to blackboard. **Do NOT clear your Jupyter notebook cell outputs that contain your results.**

Academic Integrity

Honor Code: Students may discuss the solutions at a high level with one another. Note that each student must write down their solutions/code independently to show they understand the solution well enough in order to reconstruct it by themselves. Students **should clearly mention the names of all the other students who were part of their discussion group and specify each person's contribution.**

Important: Using code or solutions obtained from the web is considered an honor code violation. You may use printed or online sources for understanding how to use Python functions but you may not search for code or theoretical solutions. We check all the submissions for plagiarism. Please include the following text excerpt in your submission zip file (in the main folder, next to folders `q1` and `q2`):

```
I [insert your name here], affirm that this is my own work,
I attributed where I used the work of others, I did not fa-
cilitate academic dishonesty for myself or others, and I used
only authorized resources for this assignment, per the GW Code
of Academic Integrity.  If I failed to comply with this sta-
tement, I understand consequences will follow my actions.  Con-
sequences may range from receiving a zero on this assignment
to expulsion from the university and may include a transcript
notation.
```

1 Recommendation Systems (5 points)

Note: Please use native Python libraries like numpy/pandas/etc to solve this problem. Spark is not required. If you run into a memory error when doing large matrix operations, please make sure you are using 64-bit Python instead of 32-bit (which has a 4GB memory limit).

Consider a ratings matrix R where each row corresponds to a user and each column corresponds to an item. If user i likes item j, then $R_{i,j} = 1$, otherwise $R_{i,j} = 0$. Also assume we have m users and n items, so matrix R is $m \times n$. Let's define a matrix P , $m \times m$, as a diagonal matrix whose i-th diagonal element is the number of items that user i likes. Similarly, a matrix Q, $n \times n$, is a diagonal matrix whose i-th diagonal element is the number of users that liked item i. See matrices below for an example.

$$R = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad P = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Q = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

(a) [0.25 pt]

Define the non-normalized user similarity matrix $T = RR^T$ (multiplication of R and transposed R). Explain the meaning of T_{ii} and T_{ij} $(i \neq j)$.

Cosine Similarity: Recall that the cosine similarity of two vectors u and v is defined as:

$$cos - sim(u, v) = \frac{u \cdot v}{||u||||v||}$$

(b) [0.5 points]

Let's define the *item similarity matrix*, S_I , $n \times n$, such that the element in row i and column j is the cosine similarity of item i and item j which correspond to column i and column j of the matrix R. Convince yourself (and the grader!) that $S_I = Q^{-1/2}R^TRQ^{-1/2}$, where $Q^{-1/2}$ is defined by $Q^{-1/2} = 1/\sqrt{Q}$ for all nonzero entries of the matrix, and 0 at all other positions. You can use the provided example matrix to show why this is true.

Note: We use $Q^{1/2}$ notation to refer to the **element-wise** square root of the matrix Q.

Repeat the same question for *user similarity matrix*, S_U where the element in row i and column j is the cosine similarity of user i and user j which correspond to row i and row j of the matrix R. Show why $S_U = P^{-1/2}RR^TP^{-1/2}$.

(c) [0.5 points]

The recommendation method using user-user collaborative filtering for user u, can be described as follows: for all items s, compute $r_{u,s} = \sum_{x \in users} cos - sim(x, u) * R_{xs}$ and recommend the k items for which $r_{u,s}$ is the largest.

Similarly, the recommendation method using item-item collaborative filtering for user u can be described as follows: for all items s, compute $r_{u,s} = \sum_{x \in items} R_{ux} * cos - sim(x, s)$ and recommend the k items for which $r_{u,s}$ is the largest.

Let's define the recommendation matrix, Γ, $m \times n$, such that $\Gamma(i, j) = r_{i,j}$.

Similar to part [b], elaborate why:

- for the item-item case, $\Gamma = RQ^{-1/2}R^T RQ^{-1/2}$.
- for the user-user case, $\Gamma = P^{-1/2}RR^T P^{-1/2}R$.

(d) [3.75 points]

In this question you will apply these methods to a real dataset. The data contains information about TV shows. More precisely, for 9985 users and 563 popular TV shows, we know if a given user watched a given show over a 3 month period.

Use the dataset from `q1/data` within the bundle for this problem. The folder contains:

- `user-shows.txt`: This is the ratings matrix R, where each row corresponds to a user and each column corresponds to a TV show. $R_{ij} = 1$ if user i watched the show j over a period of three months. The columns are separated by a space.
- `shows.txt`: This is a file containing the titles of the TV shows, in the same order as the columns of R.

We will compare the user-user and item-item collaborative filtering recommendations for the 500th user of the dataset. Let's call him Alex. (i.e. with Python's 0-based indexing, Alex=users[499].)

In order to do so, we have erased the first 100 entries of Alex's row in the matrix, and replaced them by 0s. This means that we don't know which of the first 100 shows Alex has watched. Based on Alex's behaviour on the other shows, we will give Alex recommendations on the first 100 shows. We will then see if our recommendations match what Alex had in fact watched.

- Compute the matrices P and Q.
- Using the formulas in part (c), compute Γ for the user-user collaborative filtering. Let S denote the set of the first 100 shows (the first 100 columns of the matrix). From all the TV shows in S, which are the five that have the highest similarity scores for Alex? In case of ties of similarity scores between two shows, choose the one with smaller index. Do not write the index of the TV shows, write their names using the file `shows.txt`.
- Compute the matrix Γ for the movie-movie collaborative filtering. From all the TV shows in S, which are the five that have the highest similarity scores for Alex? In case of ties between two shows, choose the one with smaller index.

For sanity check, your highest similarity score for user-user collaborative filtering should be above 900, and your highest similarity score for movie-movie filtering should be above 31.

What to submit

Please submit the following:

1. Interpretation of T_{ii} and T_{ij} [1(a)]
2. Explanation of equations for S_I and S_U [1(b)]
3. Explanation of Γ for both user-user and item-item filtering. [1(c)]
4. The answer to this question should include the followings: [1(d)]
   - The names of five TV shows that have the highest similarity scores for Alex for the user-user collaborative filtering (no need to report the similarity scores)
   - The names of five TV shows that have the highest similarity scores for Alex for the item-item collaborative filtering (no need to report the similarity scores)
   - Include your jupyter notebook in the submission (do not clear the cell outputs). Make sure you add plenty of comments to your code.

2 Implementing PageRank (5 points)

In this problem, you will learn how to implement the PageRank algorithm. You will be experimenting with a small randomly generated graph (assume that the graph has no dead-ends) provided in `graph-full.txt`.

There are 100 nodes (n = 100) in the small graph and 1000 nodes (n = 1000) in the full graph, and m = 8192 edges, 1000 of which form a directed cycle (through all the nodes) which ensures that the graph is connected. It is easy to see that the existence of such a cycle ensures that there are no dead ends in the graph. There may be multiple directed edges between a pair of nodes, and your solution should treat them as the same edge. The first column in `graph-full.txt` refers to the source node, and the second column refers to the destination node.

Note: Students who are enrolled as CSCI 6444 students must implement their solution to this question using Spark. However, you can still get half the credit if you use regular Python programming (2.5 points). CSCI 4907 students can implement the solution using regular Python programming, and can get up to 2.5 extra credit points if they implement it with Spark. For those who will be programming in Spark, note that the general computation should be done in Spark, and you may also include numpy operations whenever needed. You will not receive any credit if your answers to the questions are correct, but somehow your implementation does not produce those answers.

Spark implementation hint: You may choose to store the PageRank vector r either in memory or as an RDD. Only the matrix M of links is too large to store in memory, and you are allowed to store matrix M in an RDD. e.g. data = sc.textFile("graph-full.txt"). On an actual cluster, an RDD is partitioned across the nodes of the cluster. However, you cannot then $M =$ data.collect() which fetches the entire RDD to a single machine at the driver node and stores it as an array locally.

PageRank Implementation

Assume the directed graph $G = (V, E)$ has n nodes (numbered 1, 2,...,n) and m edges, all nodes have positive out-degree, and $M = [M_{ji}]_{n\times n}$ is an $n \times n$ matrix as defined in class such that for any $j, i \in [1, n]$:

$$M_{ji} = \begin{cases} \frac{1}{deg(i)} & \text{if}(i \to j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Here, deg(i) is the number of outgoing edges of node i in G. If there are multiple edges in the same direction between two nodes, treat them as a single edge. By the definition of PageRank, assuming $1 - \beta$ to be the teleport probability, and denoting the PageRank vector by the column vector r, we have the following equation:

$$r = \frac{1-\beta}{n}1 + \beta M r \tag{1}$$

Based on this equation, the iterative procedure to compute PageRank works as follows:

1. Initialize: $r^{(0)} = \frac{1}{n}1$
2. For i from 1 to k, iterate: $r^{(i)} = \frac{1-\beta}{n}1 + \beta M r$

Run the aforementioned iterative process in Spark for 40 iterations (assuming $\beta = 0.8$) and obtain the PageRank vector r. The matrix M can be large and should be processed as an RDD in your solution. Compute the PageRank scores and report the node id for the following using `graph-full.txt`:

- List the top 5 node ids with the highest PageRank scores.
- List the bottom 5 node ids with the lowest PageRank scores.

For a sanity check, we have provided a smaller dataset (`graph-small.txt`). In that dataset, the top node has id 53 with value 0.036. Note that the `graph-small.txt` dataset is only provided for sanity check purpose. Your write-up should include results obtained using `graph-full.txt`. Some key spark functions that may be relevant to your implementation are the following: `map()`, `distinct()`, `groupByKey()`, `cache()`, `count()`, `flatMap()`, `reduceByKey()`.

What to submit

1. List 5 node ids with the highest and least PageRank scores using `graph-full.txt`
2. Upload the code. Make sure you write comments and elaborate on the process. Remember not to clear the cell outputs from your jupyter notebook.