Northeastern University, Khoury College of Computer Science

CS 6220 Data Mining - Assignment 3

Name: Xinyue Han

Git Username: aiC0ld

Github repo link:
https://github.com/aiC0ld/CS6220-DataMining/tree/main

E-mail: han.xinyue@northeastern.edu

Map Reduce in Spark

Write a Spark program that implements a simple “People You Might Know” social network friendship recommendation algorithm. The key idea is that if two people have a lot of mutual friends, then the system should recommend that they connect with each other.

Data

- The data file is soc-LiveJournal1Adj.txt in the homework 3 folder.
- The file contains the adjacency list and has multiple lines in the following format: <User><TAB><Friends>
- Here, <User> is a unique integer ID corresponding to a unique user and <Friends> is a comma separated list of unique IDs corresponding to the friends of the user with the unique ID <User>. Note that the friendships are mutual (i.e., edges are undirected): if A is friend with B then B is also friend with A. The data provided is consistent with that rule as there is an explicit entry for each side of each edge.

Algorithm

Let us use a simple algorithm such that, for each user U, the algorithm recommends N = 10 users who are not already friends with U, but have the most number of mutual friends in common with U.

Output

- The output should contain one line per user in the following format: <User><TAB><Recommendations>
- Here, <User> is a unique ID corresponding to a user and <Recommendations> is a comma separated list of unique IDs corresponding to the algorithm's recommendation of people that <User> might know, ordered in decreasing number of mutual friends.
- Note: The exact number of recommendations per user could be less than 10. If a user has less than 10 second-degree friends, output all of them in decreasing order of the number of mutual friends. If a user has no friends, you can provide an empty list of recommendations. If there are recommended users with the same number of mutual friends, then output those user IDs in numerically ascending order.

Pipeline Sketch

Please provide a description of how you used Spark to solve this problem. Don't write more than 3 to 4 sentences for this: we only want a very high-level description of your strategy to tackle this problem.

Tips

Use Google Colab to use Spark seamlessly, e.g., copy and adapt the setup cells from the Colab done in class in the second lecture.

- Before submitting a complete application to Spark, you may go line by line, checking the outputs of each step. Command .take(X) should be helpful, if you want to check the first X elements in the RDD.
- For sanity check, your top 10 recommendations for user ID 11 should be: 27552, 7785, 27573, 27574, 27589, 27590, 27600, 27617, 27620, 27667.
- The execution may take a while. Our implementations took around 10 minutes.

Submission Instructions

- Submit your code, output file as a single file (call it output.txt) and PDF write-up via Gradescope before 4:30pm Monday, October 7, 2024.
- Code legibility is part of our grading criterion, so please make sure it's readable.

- Include a diagram of your pipeline description in your writeup.
- Include in your writeup the recommendations for the users with following user IDs: 924, 8941, 8942, 9019, 9020, 9021, 9022, 9990, 9992, 9993

Answer:

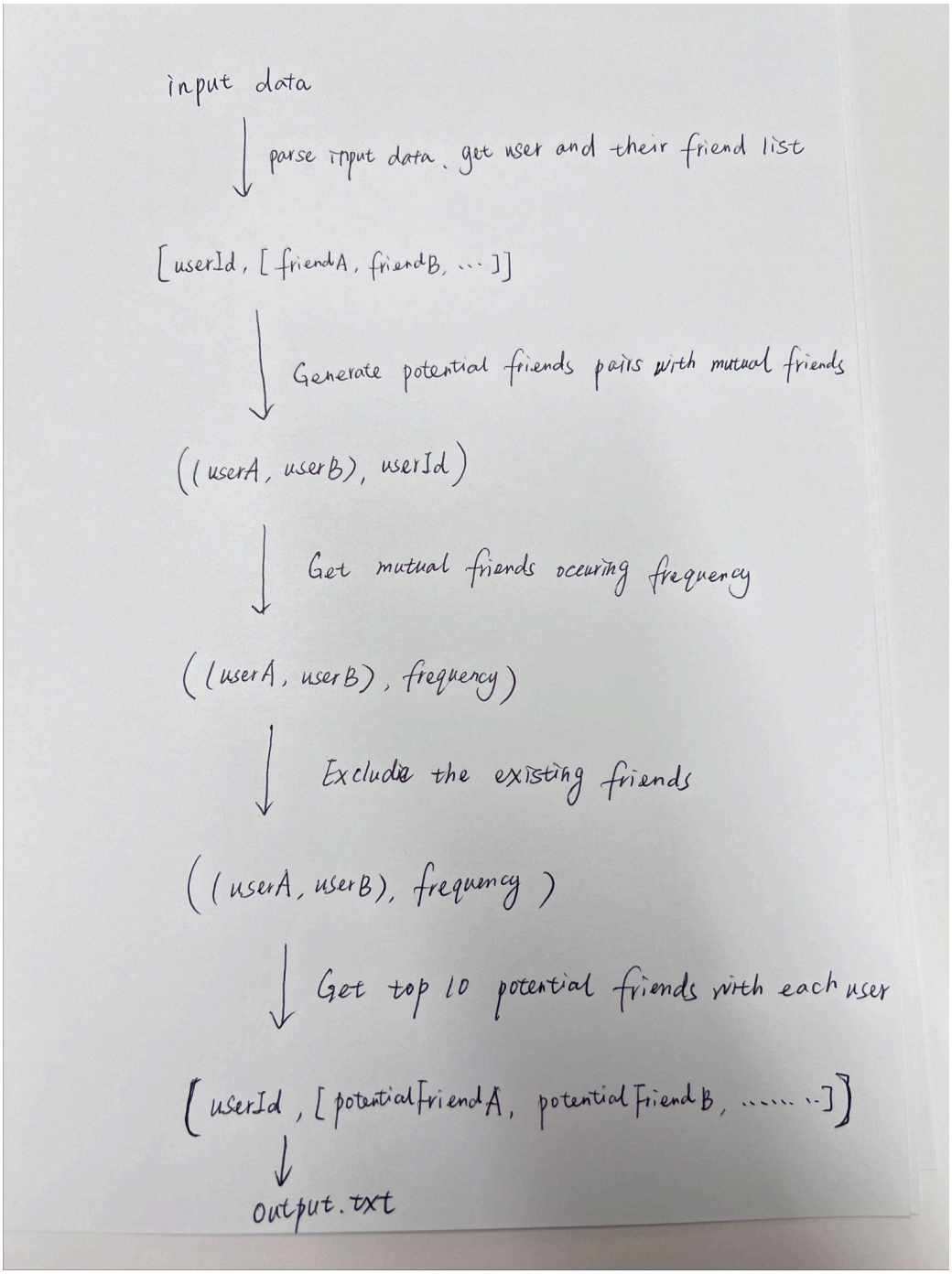
1. Recommendations for the required users

   ```
   User ID 924: 439, 2409, 6995, 11860, 15416, 43748, 45881
   User ID 9020: 9021, 9016, 9017, 9022, 317, 9023
   User ID 9019: 9022, 317, 9023
   User ID 9993: 9991, 13134, 13478, 13877, 34299, 34485, 34642, 37941
   User ID 9022: 9019, 9020, 9021, 317, 9016, 9017, 9023
   User ID 8941: 8943, 8944, 8940
   User ID 9992: 9987, 9989, 35667, 9991
   User ID 9021: 9020, 9016, 9017, 9022, 317, 9023
   User ID 9990: 13134, 13478, 13877, 34299, 34485, 34642, 37941
   User ID 11: 27552, 7785, 27573, 27574, 27589, 27590, 27600, 27617, 27620, 27667
   User ID 8942: 8939, 8940, 8943, 8944
   ```

2. Pipeline sketch and description of how I used Spark to solve this problem
   a. Description

      First, parse the input data to get each user and their direct friend list. For each user, get potential friend pairs with mutual friends from their direct friend list. Then get the frequency of each potential mutual friend pair. Exclude existing direct friends. Finally, map the potential friends with frequency by each user, sort them by the frequency, and then get the top 10 potential friends for each user.

   b. Pipeline sketch (next page)


input data
parse input data, get user and their friend list
[userId, [friendA, friendB, ...]]
Generate potential friends pairs with mutual friends
((userA, userB), userId)
Get mutual friends occuring frequency
((userA, userB), frequency)
Exclude the existing friends
((userA, userB), frequency)
Get top 10 potential friends with each user
[userId, [potentialFriendA, potentialFriendB, ...]]
output.txt