CS550: Massive Data Mining and Learning Spring 2020
Problem Set 1
Due 11:59pm Thursday, March 5, 2020

Only one late period is allowed for this homework (11:59pm Friday 3/6)

General Instructions

Submission instructions: These questions require thought but do not require long answers. Please be as concise as possible. You should submit your answers as a writeup in PDF format, for those questions that require coding, write your code for a question in a single source code file, and name the file as the question number (e.g., question_1.java or question_1.py), finally, put your PDF answer file and all the code files in a folder named as your NetID (e.g., ab123), compress the folder as a zip file (e.g., ab123.zip), and submit the zip file via Sakai.

For the answer writeup PDF file, we have provided both a word template and a latex template for you, after you finished the writing, save the file as a PDF file, and submit both the original file (word or latex) and the PDF file.

Questions

1. Map-Reduce (35 pts)

Write a MapReduce program in Hadoop 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.

Input: Use the provided input file hw1q1.zip.

The input 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 largest number of mutual friends in common with U.

Output: The output should contain one line per user in the following format:

<User><TAB><Recommendations>

where <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 by decreasing number of mutual friends. Even if a user has fewer 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 multiple users with the same number of mutual friends, ties are broken by ordering them in a numerically ascending order of their user IDs.

Also, please provide a description of how you are going to use MapReduce jobs to solve this problem. We only need a very high-level description of your strategy to tackle this problem.

Note: It is possible to solve this question with a single MapReduce job. But if your solution requires multiple MapReduce jobs, then that is fine too.

What to submit:

(i) The source code as a single source code file named as the question number (e.g., question_1.java).

(ii) Include in your writeup a short paragraph describing your algorithm to tackle this problem.

(iii) Include in your writeup the recommendations for the users with following user IDs:
924, 8941, 8942, 9019, 9020, 9021, 9022, 9990, 9992, 9993.

2. Association Rules (35 pts)

Association Rules are frequently used for Market Basket Analysis (MBA) by retailers to understand the purchase behavior of their customers. This information can be then used for many different purposes such as cross-selling and up-selling of products, sales promotions, loyalty programs, store design, discount plans and many others.

Evaluation of item sets: Once you have found the frequent itemsets of a dataset, you need to choose a subset of them as your recommendations. Commonly used metrics for measuring significance and interest for selecting rules for recommendations are:

2a. **Confidence** (denoted as conf($A \rightarrow B$)): Confidence is defined as the probability of occurrence of B in the basket if the basket already contains A:

$$\text{conf}(A \rightarrow B) = \Pr(B|A),$$

where Pr(B|A) is the conditional probability of finding item set B given that item set A is present.

2b. **Lift** (denoted as lift(A ⟶ B)): Lift measures how much more "A and B occur together" than "what would be expected if A and B were statistically independent":

$$lift(A \rightarrow B) = \frac{conf(A \rightarrow B)}{S(B)}$$

where $S(B) = \frac{Support(B)}{N}$ and N is the total number of transactions (baskets).

3. **Conviction** (denoted as conv(A⟶B)): it compares the "probability that A appears without B if they were independent" with the "actual frequency of the appearance of A without B":

$$conv(A \rightarrow B) = \frac{1 - S(B)}{1 - conf(A \rightarrow B)}$$

(a) [5 pts]

A drawback of using confidence is that it ignores Pr(B). Why is this a drawback? Explain why lift and conviction do not suffer from this drawback?

(b) [5 pts]

A measure is symmetrical if measure(A ⟶ B) = measure(B ⟶ A). Which of the measures presented here are symmetrical? For each measure, please provide either a proof that the measure is symmetrical, or a counterexample that shows the measure is not symmetrical.

(c) [5 pts]

A measure is desirable if its value is maximal for rules that hold 100% of the time (such rules are called perfect implications). This makes it easy to identify the best rules. Which of the above measures have this property? Explain why.

Product Recommendations: The action or practice of selling additional products or services to existing customers is called cross-selling. Giving product recommendation is one of the examples of cross-selling that are frequently used by online retailers. One simple method to give product recommendations is to recommend products that are frequently browsed together by the customers.

Suppose we want to recommend new products to the customer based on the products they have already browsed on the online website. Write a program using the A-priori algorithm to

find products which are frequently browsed together. Fix the support to s = 100 (i.e. product pairs need to occur together at least 100 times to be considered frequent) and find itemsets of size 2 and 3.

Use the provided browsing behavior dataset browsing.txt. Each line represents a browsing session of a customer. On each line, each string of 8 characters represents the id of an item browsed during that session. The items are separated by spaces.

Note: for the following questions (d) and (e), the writeup will require a specific rule ordering but the program need not sort the output.

(d) [10pts]

Identify pairs of items (X, Y) such that the support of {X, Y} is at least 100. For all such pairs, compute the ***confidence*** scores of the corresponding association rules: $X \Rightarrow Y$, $Y \Rightarrow X$. Sort the rules in decreasing order of confidence scores and list the top 5 rules in the writeup. Break ties, if any, by lexicographically increasing order on the left hand side of the rule.

(e) [10pts]

Identify item triples (X, Y, Z) such that the support of {X, Y, Z} is at least 100. For all such triples, compute the ***confidence*** scores of the corresponding association rules: $(X, Y) \Rightarrow Z$, $(X, Z) \Rightarrow Y$, and $(Y, Z) \Rightarrow X$. Sort the rules in decreasing order of confidence scores and list the top 5 rules in the writeup. Order the left-hand-side pair lexicographically and break ties, if any, by lexicographical order of the first then the second item in the pair.

What to submit:

Include your properly named code file (e.g., question_2.java or question_2.py), and include the answers to the following questions in your writeup:

(i) Explanation for 2(a).

(ii) Proofs and/or counterexamples for 2(b).

(iii) Explanation for 2(c).

(iv) Top 5 rules with confidence scores for 2(d).

(v) Top 5 rules with confidence scores for 2(e).

3. Locality-Sensitive Hashing (30 pts)

When simulating a random permutation of rows, as described in Sec 3.3.5 of MMDS textbook, we could save a lot of time if we restricted our attention to a randomly chosen k of the n rows, rather than hashing all the row numbers. The downside of doing so is that if none of the k rows contains a 1 in a certain column, then the result of the min-hashing is "don't know," i.e., we get no row number as a min-hash value. It would be a mistake to assume that two columns that both min-hash to "don't know" are likely to be similar. However, if the probability of getting "don't know" as a min-hash value is small, we can tolerate the situation, and simply ignore such min-hash values when computing the fraction of min-hashes in which two columns agree.

(a) [10 pts]

Suppose a column has m 1's and therefore (n-m) 0's. Prove that the probability we get "don't know" as the min-hash value for this column is at most $(\frac{n-k}{n})^m$.

(b) [10 pts]

Suppose we want the probability of "don't know" to be at most e^{-10}. Assuming n and m are both very large (but n is much larger than m or k), give a simple approximation to the smallest value of k that will assure this probability is at most e^{-10}. Hints: (1) You can use $(\frac{n-k}{n})^m$ as the exact value of the probability of "don't know." (2) Remember that for large x, $(1-\frac{1}{x})^x \approx 1/e$.

(c) [10 pts]

Note: This question should be considered separate from the previous two parts, in that we are no longer restricting our attention to a randomly chosen subset of the rows.

When min-hashing, one might expect that we could estimate the Jaccard similarity without using all possible permutations of rows. For example, we could only allow cyclic permutations i.e., start at a randomly chosen row r, which becomes the first in the order, followed by rows r+1, r+2, and so on, down to the last row, and then continuing with the first row, second row, and so on, down to row r−1. There are only n such permutations if there are n rows. However, these permutations are not sufficient to estimate the Jaccard similarity correctly.

Give an example of two columns such that the probability (over cyclic permutations only) that their min-hash values agree is not the same as their Jaccard similarity. In your answer, please provide (a) an example of a matrix with two columns (let the two columns correspond to sets denoted by S1 and S2) (b) the Jaccard similarity of S1 and S2, and (c) the probability that a random cyclic permutation yields the same min-hash value for both S1 and S2.

What to submit:

Include the following in your writeup:

(i) Proof for 3(a)

(ii) Derivation and final answer for 3(b)

(iii) Example for 3(c)