CSE547: Machine Learning for Big Data **Spring 2020**

Problem Set 1

Please read the homework submission policies.

Assignment Submission All students should submit their assignments electronically via GradeScope. No handwritten work will be accepted. Math formulas **must** be typeset using LaTeX or other word processing software that supports mathematical symbols (E.g. Google Docs, Microsoft Word). Simply sign up on Gradescope and use the course code MP8KGN. Please use your UW NetID if possible.

For the non-coding component of the homework, you should upload a PDF rather than submitting as images. We will use Gradescope for the submission of code as well. Please make sure to tag each part correctly on Gradescope so it is easier for us to grade. There will be a small point deduction for each mistagged page and for each question that includes code. Put all the code for a single question into a single file and upload it. Only files in text format (e.g. .txt, .py, .java) will be accepted. **There will be no credit for coding questions without submitted code on Gradescope, or for submitting it after the deadline**, so please remember to submit your code.

Late Day Policy All students will be given two no-questions-asked late periods, but only one late period can be used per homework and cannot be used for project deliverables. A late-period lasts 48 hours from the original deadline (so if an assignment is due on Thursday at 11:59 pm, the late period goes to the Saturday at 11:59pm Pacific Time).

Academic Integrity We take academic integrity extremely seriously. We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions and the code independently. In addition, each student should write down the set of people whom they interacted with.

Discussion Group (People with whom you discussed ideas used in your answers):

On-line or hardcopy documents used as part of your answers:

I acknowledge and accept the Academic Integrity clause.

*(Signed)*_____

1 Spark (35 pts)

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:

- Associated data file is soc-LiveJournal1Adj.txt in `q1/data`.
- 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>`

  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 in decreasing number of mutual friends.
- Even 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:

- Before submitting a complete application to Spark, you may use the Shell to 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 default memory assigned to the Spark runtime may not be enough to process this data file, depending on how you write your algorithm. If your Spark job fails with a message starting as:

```
17/12/28 10:50:35 INFO DAGScheduler: Job 0 failed: sortByKey at FriendsRecomScala.scala:45, took 519.084974 s
Exception in thread "main" org.apache.spark.SparkException:
Job aborted due to stage failure: Task 0 in stage 2.0 failed 1 times, most recent failure:
Lost task 0.0 in stage 2.0 (TID 4, localhost, executor driver)
```

  then you'll very likely need to increase the memory assigned to the Spark runtime. If you are running in stand-alone mode (i.e. you did not setup a Spark cluster), use `--driver-memory 8G` to set the runtime memory to 8GB. If you are running on a Spark cluster, use `--executor-memory 8G` to set the memory to 8GB.

What to submit

(1) Upload your code to Gradescope.

(2) Include in your writeup a short paragraph sketching your spark pipeline.

(3) 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 (45 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:

1. **Confidence** (denoted as $\text{conf}(A \to B)$): *Confidence* is defined as the probability of occurrence of B in the basket if the basket already contains A:

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

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

2. **Lift** (denoted as $\text{lift}(A \to B)$): *Lift* measures how much more "A and B occur together" than "what would be expected if A and B were statistically independent":

$$\text{lift}(A \to B) = \frac{\text{conf}(A \to B)}{S(B)},$$

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

3. **Conviction** (denoted as $\text{conv}(A \to B)$): *Conviction* compares the "probability that A appears without B if they were independent" with the "actual frequency of the appearance of A without B":

$$\text{conv}(A \to B) = \frac{1 - S(B)}{1 - \text{conf}(A \to B)}.$$

(a) [4pts]

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) [5pts]

A measure is *symmetrical* if $\text{measure}(A \to B) = \text{measure}(B \to A)$. 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) [6pts]

Perfect implications are rules that hold 100% of the time (or equivalently, the associated conditional probability is 1). A measure is *desirable* if it reaches its maximum achievable value for all perfect implications. This makes it easy to identify the best rules. Which of the above measures have this property? You may ignore 0/0 but not other infinity cases. Also you may find it easy to explain by an example.

Application in 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 online. 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 online browsing behavior dataset from `browsing.txt` in `q2/data`. 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 parts (d) and (e), the writeup will require a specific rule ordering but the program need not sort the output. We are not giving partial credits to coding when results are wrong. However, two sanity checks are provided and they should be helpful when you progress: (1) there are 647 frequent items after 1st pass ($|L_1| = 647$), (2) the top 5 pairs you should produce in part (d) all have confidence scores greater than 0.985. See detailed instructions below. You don't need to use Spark unless you want to.

(d) [15pts]

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) [15pts]

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: $(X, Y) \Rightarrow Z$, $(X, Z) \Rightarrow Y$, $(Y, Z) \Rightarrow X$. 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

Upload all the code to Gradescope and include the following 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 [2(d)].

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

3 Locality-Sensitive Hashing (20 pts)

When simulating a random permutation of rows, as described in **Sect. 3.3.5** of MMDS, we could save time if we restricted our attention to a randomly chosen k of the n rows, rather than hashing all n 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 minhashing is "don't know". In other words, we get no row number as the minhash value. It would be a mistake to assume that two columns that both minhash to "don't know" are likely to be similar. However, if the probability of getting "don't know" as a minhash value is small, we can tolerate the situation and simply ignore such minhash values when computing the fraction of minhashes in which two columns agree.

In part (a) we determine an upper bound on the probability of getting "don't know" as the minhash value when considering only a k-subset of the n rows, and in part (b) we use this bound to determine an appropriate choice for k, given our tolerance for this probability.

(a) [7pts]

Suppose a column has m 1's and therefore $n - m$ 0's, and we randomly choose k rows to consider when computing the minhash. Prove that the probability of getting "don't know" as the minhash value for this column is at most $(\frac{n-k}{n})^m$.

(b) [7pts]

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 ensure this probability is at most e^{-10}. Your expression should be a function of n and m. Hints: (1) Part a. (2) Remember that for any $x \in \mathbb{R}$, $1 + x \leq e^x$.

(c) [6pts]

Note: Part (c) 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 minhashing, 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 minhash 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 minhash 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) including the three requested items