Assignment 2: Frequent Itemsets

Spring 2016

100 points

Due: 11:59pm, 2/29/2016

This assignment has two parts. In part (1), you will implement the Apriori algorithm. In part (2), you will implement the SON algorithm using MapReduce and the Apriori algorithm you wrote in part (1).

Problem 1 (Apriori algorithm, 40 points)

Implement the Apriori algorithm, called "apriori.py". The program takes a file storing the baskets of items, and discovers all frequent itemsets with the support ratio being 30%. That is, if there are 10 baskets, then at least 3 of them should contain the itemsets.

<Input>

The input file is a single line of nested JSON array, within the array, each basket is represented as a JSON array of integers representing item numbers. A sample file is as follows:

[[1, 2], [1, 2, 3], [1, 3, 4], [2, 3, 4], [3, 4]] *# 5 baskets*

<Execute>

python firstname_lastname_apriori.py baskets.json

<Output> (to std out, do not save it to any files)

Print out candidates and frequent itemsets in each pass, each per line as a **SORTED** list, until C(k) or L(k) is empty. An example is as follows:

C1: [[1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12]]
L1: [[1], [2], [3], [4], [5], [6]]
C2: [[1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [2, 3], [2, 4], [2, 5], [2, 6], [3, 4], [3, 5], [3, 6], [4, 5], [4, 6], [5, 6]]
L2: [[1, 2], [1, 3], [2, 3], [2, 6], [3, 5], [4, 5]]
C3: [[1, 2, 3]]
L3: []

DO NOT print any additional information other than required above.

To enable re-using the Apriori algorithm you implemented in the next task, you **MUST** define the entry as a function described below:

def apriori(baskets, prRst):

The first input argument of this function 'basket' should be a list of integer list, containing all the baskets read from file, e.g.: [[1, 2], [1, 2, 3], [1, 3, 4], [2, 3, 4], [3, 4]].The second input argument of this function 'prRst' is a boolean, which controls whether to print out result of each pass. (In Problem 1 set it to True, while in Problem 2 set it to False)

The output (return) of this function should be a nested list, containing all the frequent itemsets found by the algorithm, as [[sorted list of frequent 1-itemsets], … , [sorted list of frequent k-itemsets]], e.g.: [[[1], [2], [3], [4], [6]], [[1, 2], [1, 3], [2, 3]]].

Format of other parts of your implementation is not strictly required, but you are strongly recommended to divide your code into function modules and comment as detailed as possible, so that partial credits could be given even if it fails in some test cases.

Problem 2 (SON algorithm in MapReduce, 60 points)

Recall that the key idea of SON algorithm is to divide baskets into chunks.

SON MapReduce algorithm proceeds in two phases "son-phase1.py" and "son-phase2.py".

In the first phase, it finds local frequent itemsets in each chunk as candidates. You MUST explicitly import and call your apriori(baskets, False) in the Map function, and use the returned local frequent itemsets to produce proper intermediate key-value pairs for Reduce function.

<Input>

Here you will be provided with a file that contains a list of JSON arrays, each line in the same format as the single line described in Problem 1 and corresponding to a chunk. A sample file is as follows:

[[1, 2], [1, 2, 3], [1, 3, 4], [2, 3, 4], [3, 4]] # line 1, chunk 1 with 5 baskets
[[1, 3, 4], [2, 3], [2], [3, 4]] # line 2, chunk 2 with 4 baskets
…

<Execute>

python firstname_lastname_son-phase1.py chunks.json

<Output>

one line per candidate in JSON array, for example:

[1]
[3]
[4]
[1,4]
[2,4]
…

Do not save this output to any files, but for the purpose of testing your code of phase 2, you may redirect it to 'phase1output.json' when executing:

python firstname_lastname_son-phase1.py chunks.json > phase1output.json

In the second phase, it finds global frequent itemsets among candidates from the previous phase.

You may assume that the support threshold ratio is still 30%.

<Input>

Input of this phase include two parts, one is chunks of baskets the same as phase 1 input, the other is the candidates found in phase 1.

You may use your phase1output.json as the input of phase 2, you may also use the sample output of phase 1 we provide as input of phase 2 so that any potential errors in phase 1 would not affect your test of phase 2.

<Execute>

python firstname_lastname_son-phase2.py chunks.json phase1output.json

<Output> (to std out, do not save it to any files)

One line per each global frequent itemset and its global count, in the following format:

[[1, 2], 18] # frequent itemset [1,2] with global count 18
[[1, 3], 20]
[[1], 31]
[[2], 22]
[[3], 22]
[[4], 12]
[[6], 15]
[[2, 3], 13]

Order of lines in output of both phases does not matter, just let the MapReduce framework do it for you.

The MapReduce framework code is the same as that in Assignment 1.

Submission Details

Submit and only submit the following 3 python scripts:

<firstname>_<lastname>_apriori.py

<firstname>_<lastname>_son-phase1.py

<firstname>_<lastname>_son-phase2.py

DO NOT make them into .zip.