Introduction to Massive Data Analysis

Term Project Report

Student ID: 105062635
Name: 吳浩寧

1. PROBLEM DESCRIPTION

我在 mapreduce 的架構下實作 SON (Savasere, Omiecinski, & Navathe) Algorithm，可以把它看作分散式的 Apriori Algorithm。

在 Apriori Algorithm 中，每個 pass 會掃過輸入資料的每一筆 transaction，計算出所有特定大小的 itemsets 數量，再剔除出現次數少與 threshold 的 itemsets。如此，第一個 pass 可以找出所有的 frequent singletons，第二個 pass 可以找出所有的 frequent pairs，依此類推。

在 SON Algorithm 中，則會將大檔案切割成數個塞得下記憶體的小檔案，分別對 N 個小檔案執行 Apriori，以取得出現次數大於 threshold/N 的 frequent itemsets，計算這些 itemsets 在所有的檔案出現的總次數，剔除出現次數少與 threshold 者，即為最後的結果。

我的測資來自 https://wiki.csc.calpoly.edu/datasets/中的 Extended BAKERY dataset，資料內容為麵包店販售的 40 種糕點與 10 種咖啡，分別有 1000、5000、20000、75000 筆 transactions，使用下列網站計算 Apriori：http://www.stahamtan.com/ expertise/business-intelligence/association-rule-mining/，以驗證自己程式的正確性。

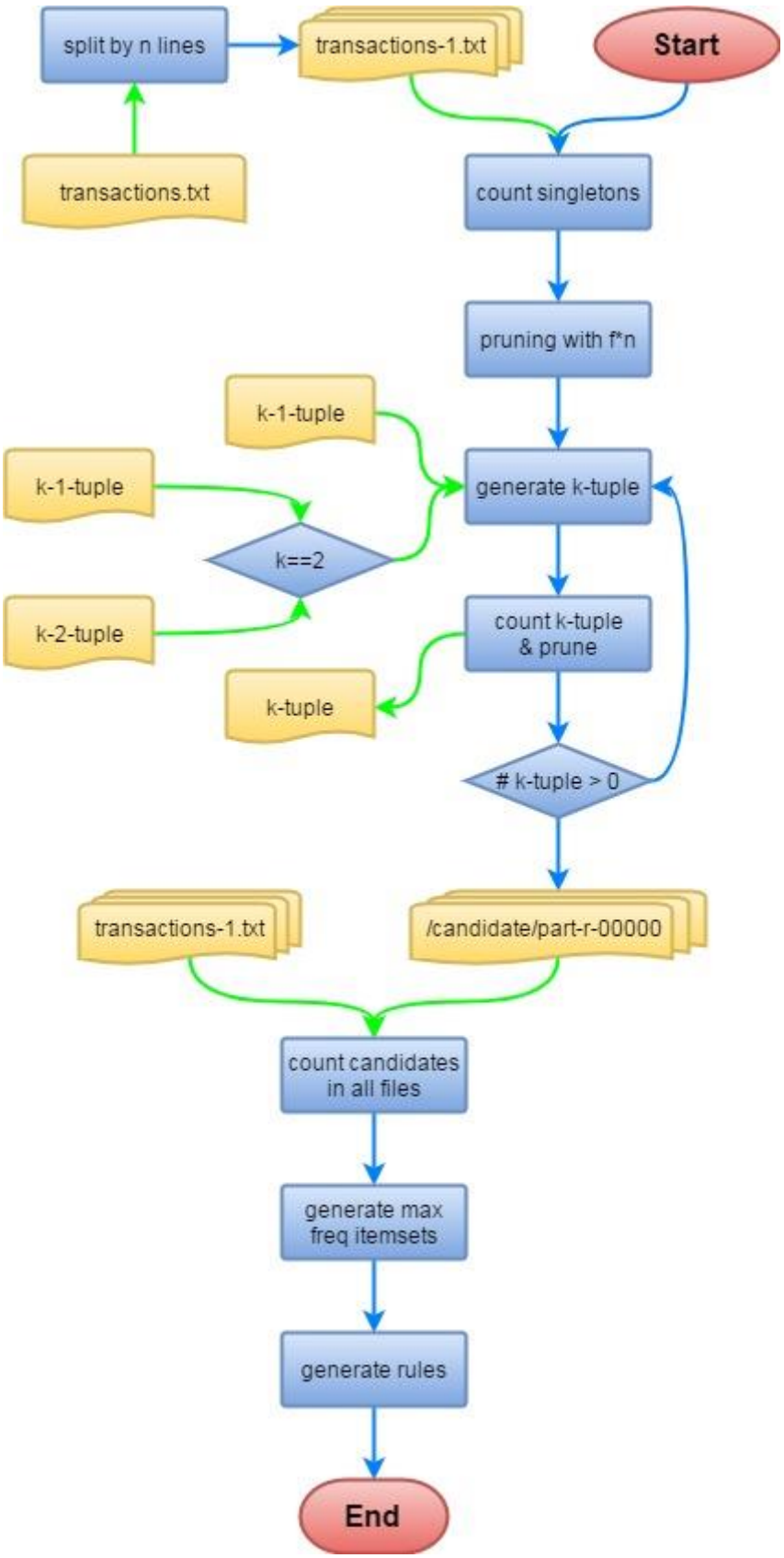
2. IMPLEMENTATION

SON <in> <out> <threshold> <filesize> <chunksize> <tuplesize>

threshold	frequent itemset 出現頻率的門檻值
chunksize	每個子檔案的大小(行數)
filesize	整個輸入檔的大小(行數)
tuplesize	最大要找出多大的 candidate k-tuple

1.) File Split

由於必須將原輸入檔案平均分配給每個 mapper，因此不能使用 Hadoop 預設的 TextInputFormat；雖然有 NLineInputFormat，可以讓每個 mapper 負責 N 行的 file split，但用這種方式產生的每個 record 為一行資料，代表每次呼叫 map()僅能處理一筆資料，無法對整個檔案統一操作。因此我另外寫了 MultiLineInputFormat 繼承 NLineInputFormat，並 override 其 RecordReader 的 nextKeyValue()方法，加入一個執行 N 次的迴圈，讓每筆 record 的 value 變為 N 行資料，key 值則變為這個 file split 的行數。


split by n lines
transactions-1.txt
Start
transactions.txt
count singletons
pruning with f*n
k-1-tuple
k-1-tuple
generate k-tuple
k==2
k-2-tuple
count k-tuple
& prune
k-tuple
k-tuple > 0
transactions-1.txt
/candidate/part-r-00000
count candidates
in all files
generate max
freq itemsets
generate rules
End

2.) Job 1

Map Function:

針對 input 檔案的其中 N 行，對其進行 Apriori 運算，將每筆 transaction 讀入 ArrayList<TreeSet<Integer>>中，同時進行 singletons 數量的計算。我使用 TreeMap、TreeSet 等資料結構而非 HashMap、HashSet 的原因是它們有排序的功能，且用紅黑樹來儲存資料，因此根據 key 值取得資料只須 O(log(n))；我將所有的 candidate itemsets 採用 TreeMap<Integer, TreeMap<String, Integer>>的格式來儲存，意義為[k, k-tuple, count]，k-tuple 中的每個元素以逗號區隔，例如：[3, “1,3,5”, 4]即代表大小 3 的 itemset {1, 3, 5}在這個子檔案裡出現了 4 次。為了進行集合的運算，會將 k-tuple 的 String 轉為 array，再轉為 TreeMap，就能用 Set 中的 add()和 containsAll()方法，分別進行聯集和判斷子集的動作。

接著利用 frequent singlestons 計算 candidate pairs，利用 frequent pairs 和 frequent singletons 計算 candidate triples，依此類推，可以利用 frequent k-1-tuples 和 frequent k-2-tuples 計算出 candidate k-tuples，每一輪必須掃過所有 transactions，以剔除出現次數未達門檻者，門檻值即為 threshold frequency 乘上這個 file split 的行數，k 會一直增加到使用者給定的值，或所有的 candidate k-tuples 都無法達到門檻時，最後將所有 candidate itemsets 傳至 reducer。

以下列指令為例，threshold 為 14*0.45=6.3，若設定 file split 為 5 行，則會將檔案分成(5,5,4)，每個檔案進行 Apriori 使用的 threshold 則為(2,2,1)

./SON /user/root/data/data.txt output/out1 0.45 14 5 5

input		output1	output2	output3
1 2	1 2 3 4	1 4	1 3	2 1
1 3	1 5	2 3	2 2	3 2
2 3	2 3 4	3 4	3 3	5 3
1 2 3	2 6	1,2 3	4 3	6 2
1 2 3	3 4 5	1,3 3	1,2 2	2,6 1
4	3 5 6	2,3 3	2,3 3	3,4 1
1 2 3	5		3,4 2	4,5 1
				3,5 1
				3,6 1
				5,6 1

Reduce Function:

將 Mapper 產生的 candidate itemsets 寫入檔案。

3.) Job 2

Map Function:

需要使用一開始所有 transactions 的輸入檔，和上一階段產生的輸出檔，因此 override setup()方法，讓每個 mapper 都先從 Hadoop filesystem 讀出 candidate itemsets 存入 TreeMap<String, Integer>中。接著計算每個 itemset 在子檔案中出現的次數，若大於 0 次則將次數傳給 reducer。

Reduce Function:

統計所有 candidate itemsets 出現的次數，並剔除未達門檻者，最後即得到真正的 frequent itemsets，在此須注意門檻值的計算，假設 threshold frequency 為 f，整個檔案有 n 行，當 n*f 為整數時，threshold 為 n*f；若 n*f 非整數，則 threshold 應為 floor(n*f+1)，因此 threshold=(int)n*(f-1)+1。

4.) Post-processing

根據前面計算出來的 frequent itemsets，推導出所有的 association rules。利用 ArrayList<TreeSet<Integer>>記錄所有的 itemsets，根據集合大小從大排到小；另外用 TreeMap<String, Integer>記錄每個 itemset 的 support。首先從 List 中剔除所有 singletons，和是別的 itemset 子集者，留下的即為 maximal frequent itemsets。對 itemset S 中的元素 i，即可得到：

rule：S-{i}→i

confidence：support(S)/support(S-{i})

interest：confidence-support({i})/n

3. RESULT

./SON /user/root/data/data.txt output/out1 0.45 14 5 5

output

----------Frequent Itemsets----------

Itemset	Count
1	7
2	8
2,3	6
3	9

----------Association Rules----------

Rule	Confidence	Interest
[3]→2	0.6666666666666666	0.09523809523809523
[2]→3	0.75	0.1071428571428571

./SON /user/root/data/75000.txt output/out1 0.02 75000 10000 5

截取 confidence、interest>0.9 者，由大排到小可以得到以下結果：

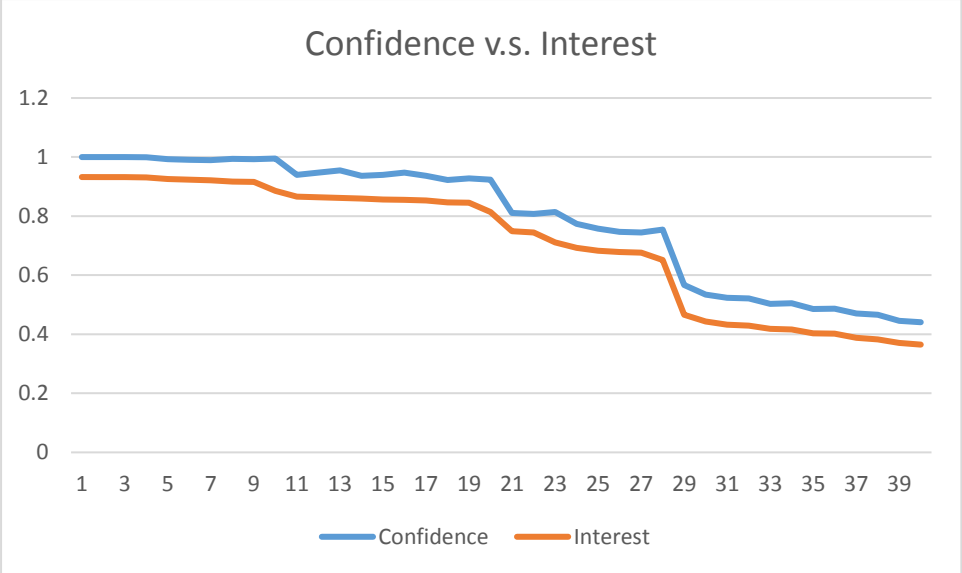
Rule	Confidence	Interest
[24,40,41,43]→23	1	0.93236
[23,24,40,43]→41	1	0.932253333
[23,24,41,43]→40	1	0.93176
[23,40,41,43]→24	0.999357326	0.931343993
[12,36,48]→31	0.992926045	0.925632712
[12,31,48]→36	0.991014121	0.923320787
[31,36,48]→12	0.98974359	0.921516923
[7,11,45]→37	0.99383009	0.91663009
[7,37,45]→11	0.992887624	0.915767624

舉第 1, 5, 8 條 rule 為例，將 ID 轉成商品名稱可以分別得到

{Lemon Cookie, Green Tea, Lemon Lemonade, Raspberry Cookie}→ Raspberry Lemonade

{Apple Tart, Apple Danish, Cherry Soda}→Apple Croissant

{Apple Pie, Hot Coffee, Coffee Éclair}→Almond Twist



可以看出在這個 dataset 上 interest 和 confidence 有正相關的關係，可能代表每樣商品被購買的機率都差不多。

4. EXPERIENCE

這次 Project 讓我了解如何自行找尋題目，並設定適當的難度，才能學到東西，又不會難到無法完成。像這次主要學到如何自訂 FileInputFormat；如何使用 Java 特有的 Object 來減輕 coding 的負擔；也注意到一些 SON algorithm 的小細節。此外網路上的資源非常多，有別人寫好的網頁板可以來驗證自己的結果，但要選到大小可以在單機上跑的 dataset 也著實花了一些時間。