INF553 Summer 2018 Assignment 2 – Ye Joo Park (USC# 1128-6851-51)

Approach to implement the algorithm

The algorithm is implemented in a similar way to the textbook’s “6.4.4 The SON Algorithm and MapReduce” section. To process each chunks, apriori is used. I have attempted to use PCY, but I wasn’t able to determine the right number of buckets.

The only difference to the textbook’s MapReduce steps is that a hashset-type data structure is used in the first reduce step instead of using (Frequent Itemset, 1) pair.

In each iteration of finding itemsets of k elements, items that are not in the frequent set k – 1 are removed from the baskets. In the code below, frequent_set variable holds the set of individual items in the frequent itemsets.

```
baskets = baskets.map(lambda basket:
basket.intersection(frequent_set))\
   .filter(lambda basket: len(basket) > 0)
```

Code in Python to remove items not in frequent set from k – 1 th iteration

To avoid redundant checks, each candidate itemset with k elements will ensure that all of its subsets of k – 1 elements are frequent itemsets. This is done by holding the frequent itemsets of k – 1 th iteration at each k th iteration and looking up for the itemset tuple.

```
subsets = itertools.combinations(combination, k - 1)
for subset in subsets:
   subset = tuple(sorted(subset))
   if subset not in self.frequent_items:
      has_monotonic_subsets = False
      break
```

Code in Python to remove items not in frequent set from k – 1 th iteration

Command line formats and examples

For Python	
Format	`spark-submit YeJoo_Park_SON.py <case number> <csv file path> <support>`
Example	`spark-submit YeJoo_Park_SON.py 1 Small1.csv 4`

For Scala

For Scala	
Format	`spark-submit --class FrequentItemsetsSON YeJoo_Park_SON.jar <case number> <csv file path> <support>`
Example	`spark-submit --class FrequentItemsetsSON YeJoo_Park_SON.jar 1 Small1.csv 4`

Execution Tables for Python

```
start_time = time.time()
son = SONAlgorithm(case_number, csv_file_path, support, app_name,
output_dir)
son.start()
print("--- %s seconds ---" % (time.time() - start_time))
```

Problem 2 (in Python)			
Case 1		Case 2	
Support Threshold	Execution Time	Support Threshold	Execution Time
120	33 secs	180	137 secs
150	24 secs	200	74 secs

Problem 3 (in Python)			
Case 1		Case 2	
Support Threshold	Execution Time	Support Threshold	Execution Time
30000	201 secs	2800	189 secs
35000	170 secs	3000	192 secs

Execution Tables for Scala

```
val startTime = System.currentTimeMillis()
// run the SON algorithm here..
val endTime = System.currentTimeMillis()
println("Time=", (endTime - startTime) / 1000)
```

Problem 2 (in Scala)			
Case 1		Case 2	
Support Threshold	Execution Time	Support Threshold	Execution Time
120	15 secs	180	371 secs
150	10 sec	200	133 secs

Problem 3 (in Scala)			
Case 1		Case 2	
Support Threshold	Execution Time	Support Threshold	Execution Time
30000	404 secs	2800	51 secs
35000	99 secs	3000	34 secs