1. Suppose that the data mining task is to cluster the following nine points (with \((x,y)\) representing location) into three clusters: \(A_1(3,10), A_2(3,5), A_3(9,4), B_1(4,8), B_2(7,5), B_3(6,4), C_1(2,2), C_2(5,9), C_3(6,8)\) Suppose initially we assign \(A_1, B_1\) and \(C_1\) as the center of each cluster, respectively. Please add a Map-reduce function for the K-means algorithm. Show the results for the first two iterations and explain how Map-reduce can help.

Answer:

Map Reduce:

Let \(K_1, K_2, K_3\) be the three centroids for the current iteration, let \(d(P, K)\) be the distance between points \(P\) and \(K\).

Map: We map each points with coordinates \(i, j\), \(P(i, j)\) to \(l\) if \(P\) is closest to centroid \(K_l\). A sample map function could be

\[
\text{map}(P) \{ \\
\quad \text{emit(index_of_closest_centroid}(K_1, K_2, K_3, P), P) \\
\}
\]

Reduce:

\[
\text{reduce(tuples)} \{ \\
\quad \text{return \{tuples.centroid_index, sum(tuples.x)/tuples.count, sum(tuples.y)/tuples.count\}}; \\
\}
\]

Execution:

Iteration 1:

Cluster 1: \(A\)
Centroid 1: \((3, 10)\)

Cluster 2: \(C, B, C, B, B, A, A\)
Centroid 2: \((5.71, 6.14)\)

Cluster 3: \(C\)
Centroid 3: \((2, 2)\)
Iteration 2:

Cluster 1: A, B, C,
Centroid 1: (4, 9)

Cluster 2: C, A, B, B, A,
Centroid 2: (6.2, 5.2)

Cluster 3: C,
Centroid 1: (2, 2)

Benefit:

The map reduce can help in the sense that all these operations can run in parallel.

2. A database has six transactions. Let min sup = 50%.

<table>
<thead>
<tr>
<th>TID</th>
<th>items_sold</th>
</tr>
</thead>
<tbody>
<tr>
<td>T001</td>
<td>A, B, C, D, E, F</td>
</tr>
<tr>
<td>T002</td>
<td>B, H, S, C, F, T</td>
</tr>
<tr>
<td>T003</td>
<td>A, U, O, F, W, D</td>
</tr>
<tr>
<td>T004</td>
<td>W, A, B, C, F, X</td>
</tr>
<tr>
<td>T005</td>
<td>W, A, C, D, F, Y</td>
</tr>
<tr>
<td>T006</td>
<td>B, C, X, E, O, Z</td>
</tr>
</tbody>
</table>

Please add a Map-reduce function for the Apriori algorithm to generate all frequent itemsets. Show the results for each step and explain how Map-reduce can help.

Answer:

Map: For each iteration, generate frequent items.

    function(doc) {
        var iteration;
        var frequent_itemset;
if(0 == iteration)
    for_each(trans = doc.transactions)
        for_each(item = trans.items)
            frequent_itemset.add(item);

else{
   // the function below take each two frequent_itemset, generate the union of them and add in the set.
   frequent_itemset = set( Cartesian_union(frequent_itemset) );
}

for_each(trans in doc.transactions){
    for_each(item in frequent_itemset){
        if(trans.contains(item )){
            emit(item, 1);
        }
    }
}

Reduce:  Count all the emitted items, compare with the support threshold.
reduce(keys, values, reducer){
    var count = _count;
    if(count / doc.transactions.length > doc.support)
        return (c, value)
}

\(L_1\) (frequency >= 3):

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
</tr>
<tr>
<td>F</td>
<td>5</td>
</tr>
<tr>
<td>W</td>
<td>3</td>
</tr>
</tbody>
</table>
Candidate C

AB, AC, AF, AD, AW, BC, BD, BF, BW, CD, CF, CW, DF, DW, FW.

L.

A,C 3
A,D 3
A,F 4
A,W 3
B,C 4
C,F 4
B,F 3
D,F 3
F,W 3

C:

ACF, ADF, AFW, BCF

L,

ACF 3
ADF 3
AFW 3
BCF 3

C.

None.

Benefit:
The map reduce function can help process the map reduce function in parallel.