Discovering Correlated Pattern in Big Data Using CPGrowth Algorithm

In this tutorial, we will discuss two approaches to find Correlated Patterns in big data using CPGrowth algorithm.

1. **Basic approach:** Here, we present the steps to discover Correlated Pattern using a single minimum support value
2. **Advanced approach:** Here, we generalize the basic approach by presenting the steps to discover Correlated Pattern using multiple minimum support values.

Basic approach: Executing CPGrowth on a single dataset at a particular minimum support value

Step 1: Import the CPGrowth algorithm

In [1]:
```
from PAMI.correlatedPattern.basic import CPGrowth  as alg
```

Step 2: Specify the following input parameters

In [2]:
```
inputFile = 'transactional_T10I4D100K.csv'
minAllConfCount=0.1
minimumSupportCount=100  #Users can also specify this constraint between 0 to 1.

seperator='¥t'
```

Step 3: Execute the CPGrowth algorithm

In [3]:
```
obj = alg.CPGrowth(iFile=inputFile, minSup=minimumSupportCount,  minAllConf=minAllCo
obj.startMine()              #Start the mining process
```

```
IOPub data rate exceeded.
The Jupyter server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--ServerApp.iopub_data_rate_limit`.

Current values:
ServerApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
ServerApp.rate_limit_window=3.0 (secs)
```

```
Correlated Frequent patterns were generated successfully using CorrelatedPatternGrow
th algorithm
```

Step 4: Storing the generated patterns

Step 4.1: Storing the generated patterns in a file

In [4]:
```
obj.savePatterns(outFile='frequentPatternsMinSupCount100.txt')
```

Step 4.2. Storing the generated patterns in a data frame

```
In [5]: frequentPatternsDF= obj.getPatternsAsDataFrame()
```

Step 5: Getting the statistics

Step 5.1: Total number of discovered patterns

```
In [6]: print('Total No of patterns: ' + str(len(frequentPatternsDF)))
```

```
Total No of patterns: 5758
```

Step 5.2: Runtime consumed by the mining algorithm

```
In [7]: print('Runtime: ' + str(obj.getRuntime()))
```

```
Runtime: 11.846760034561157
```

```
In [8]: ##### Step 5.3: Total Memory consumed by the mining algorithm
```

```
In [9]: print('Memory (RSS): ' + str(obj.getMemoryRSS()))
        print('Memory (USS): ' + str(obj.getMemoryUSS()))
```

```
Memory (RSS): 410480640
Memory (USS): 372371456
```