parallelFPGrowth-st

September 5, 2022

1 Discovering Frequent Patterns in Big Data Using parallelFP-Growth Algorithm

In this tutorial, we will discuss two approaches to find frequent patterns in big data using parallelFPGrowth algorithm.

1. Basic approach: Here, we present the steps to discover frequent patterns using a single minimum support value

---

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

Step 1: Import the parallelFPGrowth algorithm

```
[1]: from PAMI.frequentPattern.pyspark import parallelFPGrowth  as alg
```

Step 2: Specify the following input parameters

```
[2]: inputFile = 'transactional_T10I4D100K.csv'

minimumSupportCount=100   #Users can also specify this constraint between 0 to 1.
mumberWorkersCount=2
seperator='\t'
```

Step 3: Execute the parallelFPGrowth algorithm

```
[3]: obj = alg.parallelFPGrowth(iFile=inputFile,␣
     ↪minSup=minimumSupportCount,numWorkers=mumberWorkersCount, sep=seperator)   ␣
     ↪#initialize
     obj.startMine()              #Start the mining process
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).

22/08/29 19:01:14 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
```

Frequent patterns were generated successfully using Parallel FPGrowth algorithm

Step 4: Storing the generated patterns

Step 4.1: Storing the generated patterns in a file

```
[4]: obj.savePatterns(outFile='frequentPatternsMinSupCount1000.txt')
```

Step 4.2. Storing the generated patterns in a data frame

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

Step 5: Getting the statistics

Step 5.1: Total number of discovered patterns

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

Total No of patterns: 27532

Step 5.2: Runtime consumed by the mining algorithm

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

Runtime: 23.49443221092224

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

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

Memory (RSS): 132694016
Memory (USS): 118325248