FAE

July 1, 2022

1 Discovering Frequent Patterns in Big Data Using FAE Algorithm

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

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

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

Step 1: Import the FAE algorithm

[1]:
```
from PAMI.frequentPattern.topk import FAE  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.

seperator='\t'
```

Step 3: Execute the FAE algorithm

[3]:
```
obj = alg.FAE(iFile=inputFile, k=minimumSupportCount, sep=seperator)
#initialize
obj.startMine()            #Start the mining process
```

```
100
FAE has successfully generated top-k frequent patterns
```

Step 4: Storing the generated patterns

Step 4.1: Storing the generated patterns in a file

[4]:
```
obj.savePatterns(outFile='frequentPatternsMinSupCount100.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: 100
```

Step 5.2: Runtime consumed by the mining algorithm

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

```
Runtime: 2.242154121398926
```

[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): 214978560
Memory (USS): 176656384
```

1.2 Advanced approach: Executing FAE on a single dataset at multiple minimum support values

Step 1: Import the FAE algorithm

[10]:
```
from PAMI.frequentPattern.topk import FAE  as alg
```

Step 2: Specify the following input parameters

[11]:
```
inputFile = 'transactional_T10I4D100K.csv'

minimumSupportCountList = [100, 150, 200, 250, 300]
#Users can also specify this constraint between 0 to 1. E.g., minSupList = [0.
 ↪005, 0.006, 0.007, 0.008, 0.009]

seperator='\t'
```

Step 3: Initalize Data Frame to save values

[12]:
```
import pandas as pd
result = pd.DataFrame(columns=['algorithm', 'minSup', 'patterns', 'runtime',␣
 ↪'memory'])
```

Step 4: Execute the FAE algorithm using a for loop

[13]:
```
algorithm = 'FAE'
for minSupCount in minimumSupportCountList:
    obj = alg.FAE('transactional_T10I4D100K.csv', k=minSupCount, sep=seperator)
    obj.startMine()
    df = pd.DataFrame([algorithm, minSupCount, len(obj.getPatterns()), obj.
 ↪getRuntime(), obj.getMemoryRSS()], index=result.columns).T
    result = result.append(df, ignore_index=True)
```

```
100
FAE has successfully generated top-k frequent patterns
```

```
/tmp/ipykernel_644158/2222870977.py:6: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  result = result.append(df, ignore_index=True)
```

```
150
FAE has successfully generated top-k frequent patterns
200
FAE has successfully generated top-k frequent patterns
250
FAE has successfully generated top-k frequent patterns
300
FAE has successfully generated top-k frequent patterns
```

Step 5: Ploting the graphs

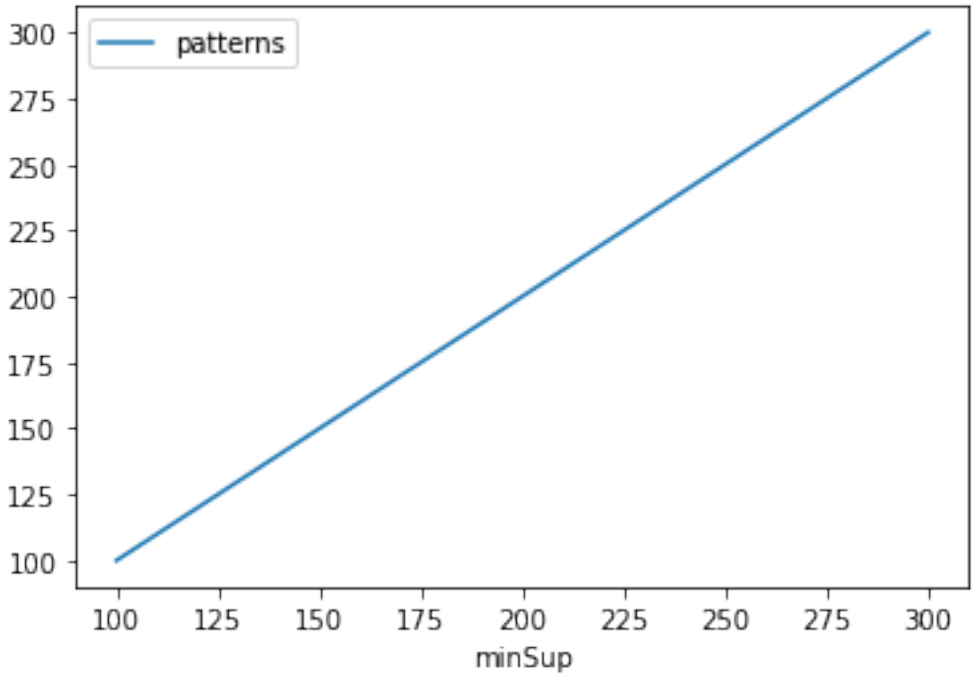
Step 5.1 Importing the plot library

[14]:
```
from PAMI.extras.graph import plotLineGraphsFromDataFrame as plt
```

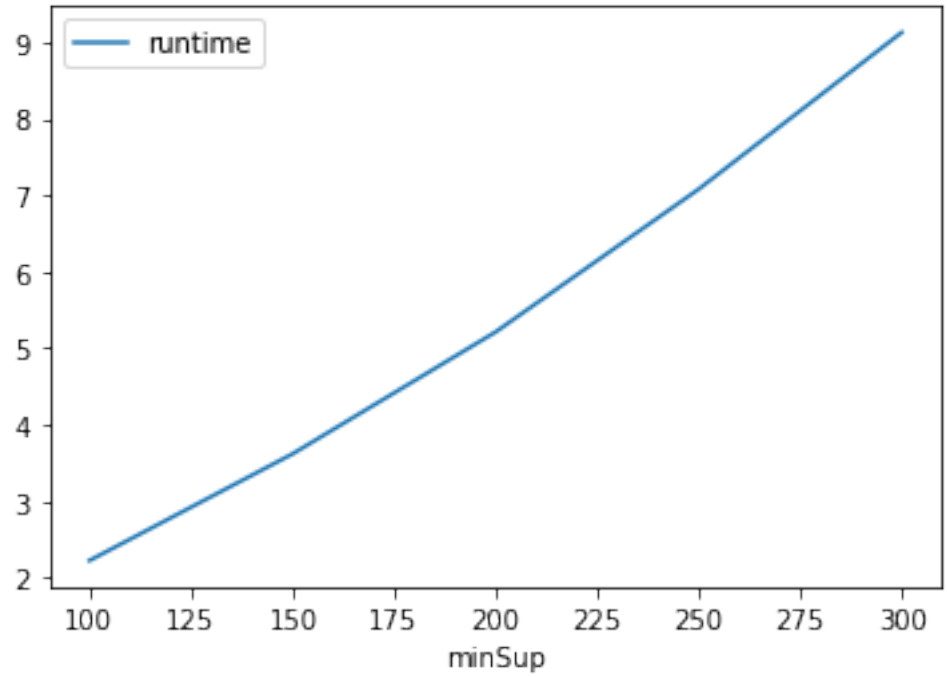
Step 5.2. Plotting the number of patterns

[15]:
```
ab = plt.plotGraphsFromDataFrame(result)
ab.plotGraphsFromDataFrame()
```

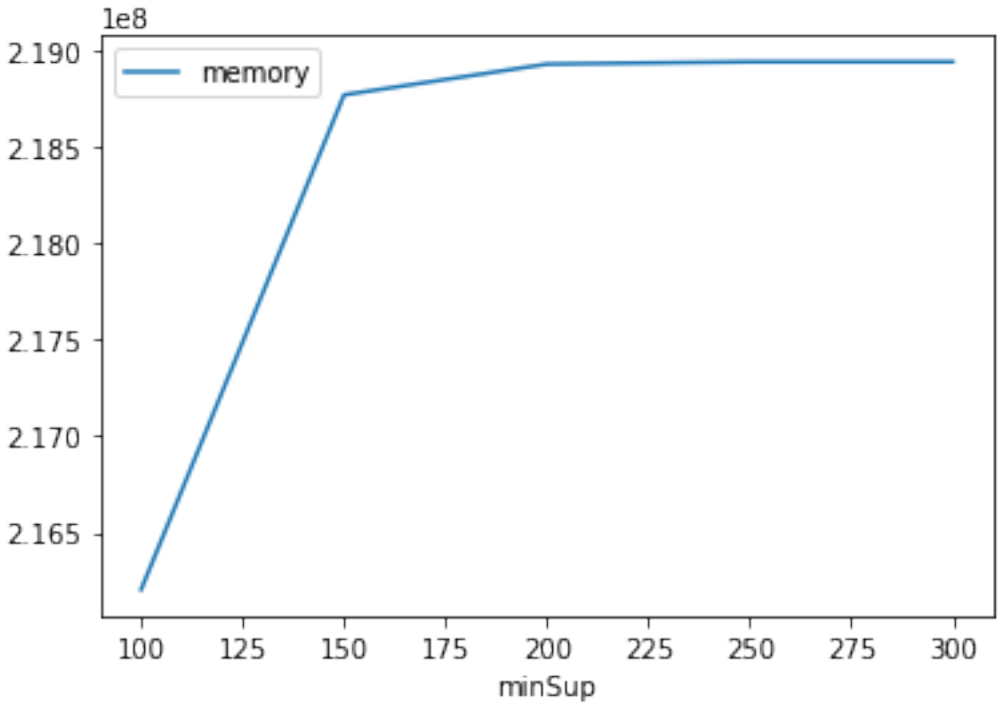
```
/home/jupyterHub/anaconda3/envs/jupyterHub/lib/python3.10/site-
packages/pandas/core/indexes/base.py:6982: FutureWarning: In a future version,
the Index constructor will not infer numeric dtypes when passed object-dtype
sequences (matching Series behavior)
  return Index(sequences[0], name=names)
```




```
Graph for No Of Patterns is successfully generated!
```




```
Graph for Runtime taken is successfully generated!
```




```
Graph for memory consumption is successfully generated!
```

1.2.1 Step 6: Saving the results in a latex file

[16]:
```
from PAMI.extras.graph import generateLatexFileFromDataFrame as gdf
gdf.generateLatexCode(result)
```

```
Latex files generated successfully
```