Mining Frequent Patterns in Uncertain Transactional Databases

What is frequent pattern mining?

Frequent pattern mining aims to discover all interesting patterns in a transactional database that have **support** no less than the user-specified **minimum support** (**minSup**) constraint. The **minSup** controls the minimum number of transactions that a pattern must appear in a database.

What is a uncertain transactional database?

A transactional database is a collection of transactions, where each transaction contains a transaction-identifier and a set of items with their respective uncertain value.
A hypothetical transactional database containing the items ***a, b, c, d, e, f, and g*** as shown below.

tid	Transactions
1	a(0.4) b(0.5) c(0.2) g(0.1)
2	b(0.2) c(0.3) d(0.4) e(0.2)
3	a(0.3) b(0.1) c(0.3) d(0.4)
4	a(0.2) c(0.6) d(0.2) f(0.1)
5	a(0.3) b(0.2) c(0.4) d(0.5) g(0.3)
6	c(0.2) d(0.7) e(0.34) f(0.2)
7	a(0.6) b(0.4) c(0.3) d(0.2)
8	a(0.2) e(0.2) f(0.2)
9	a(0.1) b(0.3) c(0.2) d(0.4)
10	b(0.3) c(0.2) d(0.1) e(0.6)

Note: Duplicate items must not exist in a transaction.

What is the acceptable format of a uncertain transactional databases in PAMI?

Each row in a transactional database must contain only items with their respective uncertain values. A sample transactional database, say sampleInputFile.txt, is provided below.

a(0.4) b(0.5) c(0.2) g(0.1)
b(0.2) c(0.3) d(0.4) e(0.2)
a(0.3) b(0.1) c(0.3) d(0.4)
a(0.2) c(0.6) d(0.2) f(0.1)
a(0.3) b(0.2) c(0.4) d(0.5) g(0.3)
c(0.2) d(0.7) e(0.34) f(0.2)
a(0.6) b(0.4) c(0.3) d(0.2)
a(0.2) e(0.2) f(0.2)
a(0.1) b(0.3) c(0.2) d(0.4)
b(0.3) c(0.2) d(0.1) e(0.6)

What are the input parameters?

The input parameters to a frequent pattern mining algorithm are:

- **Transactional database**
  Acceptable formats:

  > - String : E.g., 'uncertainTransactionalDatabase.txt'
  > - URL : E.g., https://u-aizu.ac.jp/~udayrage/datasets/transactionalDatabases/transactional_T10
  > - DataFrame with the header titled 'Transactions'

- **minSup**
  specified in

  > - **count (beween 0 to length of a database)** or
  > - [0, 1]

- **seperator**
  default seperator is '\t' (tab space)

How to store the output of a frequent pattern mining algorithm?

The patterns discovered by a frequent pattern mining algorithm can be saved into a file or a data frame.

How to run the frequent pattern mining algorithms in a terminal?

- Download the PAMI source code from github.
- Unzip the PAMI source code folder and enter into uncertain frequent pattern folder.
- Enter into uncertainFrequentPattern folder
- You will find folder like **basic**
- Enter into a specific folder and execute the following command on terminal.

syntax: python3 algorithmName.py `<path to the input file>` `<path to the output file>` `<minSup>` `<seperator>`

Example: python3 `PUFGrowth.py` `inputFile.txt` `outputFile.txt` 0.05 `' '`

How to execute a frequent pattern mining algorithm in a Jupyter Notebook?

Import the PAMI package executing: **pip3 install PAMI**

- Install the PAMI package from the PYPI repository by executing the following command: **pip3 install PAMI**
- Run the below sample code by making necessary changes

In [ ]:
```
import PAMI.uncertainFrequentPattern.basic.PUFGrowth as alg

iFile = 'sampleInputFile.txt'  #specify the input transactional database
minSup = 0.5                  #specify the minSup value <br>
seperator = ' '                #specify the seperator. Default seperator
oFile = 'frequentPatterns.txt'   #specify the output file name<br>

obj = alg.PUFGrowth(iFile, minSup, seperator) #initialize the algorithm <
obj.startMine()                        #start the mining process <br>
obj.savePatterns(oFile)                #store the patterns in file <br>
df = obj.getPatternsAsDataFrame()      #Get the patterns discovered into a
obj.printStats()                       #Print the statistics of mining pro
```

The frequentPatterns.txt file contains the following patterns (*format:* pattern:support):!cat frequentPatterns.txt

In [4]:
```
!cat frequentPatterns.txt
```

f 0.5
e 1.3399999999999999
b 2.0
b a 0.56
b c 0.51
a 2.099999999999996
a c 0.6100000000000001
c 2.7
d 2.9000000000000004
c d 0.8600000000000001

The dataframe containing the patterns is shown below:

In [ ]:
```
df
```

	Patterns	Support
0	f	0.50
1	e	1.34
2	b	2.00
3	b a	0.56
4	b c	0.51
5	a	2.09
6	a c	0.61
7	c	2.70
8	d	2.90
9	c d	0.86