Department of Computer Science and Engineering (Data Science)

Machine Learning – IV

Experiment 7

Name: Umang Kirit Lodaya **SAP ID: 60009200032**

Batch: D11

Aim:

The aim of this lab is to implement and comprehend the working principles of the Park-Chen-Yu (PCY) algorithm. PCY is a data mining algorithm designed to efficiently discover frequent itemsets in large datasets using a combination of hashing and counting techniques.

Theory:

Introduction to PCY Algorithm:

- The PCY algorithm, introduced by Park, Chen, and Yu, is a variant of the Apriori algorithm used for mining frequent itemsets in large databases.
- PCY employs a two-step process: first, it uses a hash function to count item pairs, and second, it filters the candidates based on their counts.
- The primary goal is to reduce the number of candidate itemsets that need to be examined, thereby improving efficiency.

Algorithm Overview:

- **Step 1: Hashing and Counting Pairs**
  - Use a hash function to hash the pairs of items and maintain a hash table to count the occurrences of each pair.
  - To efficiently use memory, the algorithm only counts pairs that hash to specific buckets (e.g., those whose hash value is a multiple of a certain number).
- **Step 2: Candidate Generation and Counting**
  - Generate candidate itemsets from the frequent itemsets found in the first step.
  - Count the occurrences of these candidate itemsets in the dataset.

- **Step 3: Frequent Itemset Filtering**
    - Eliminate candidate itemsets that do not meet the minimum support threshold.

Step-by-Step Implementation:

- **Step 1: Counting Pairs Using Hashing**
    - Implement a hash function for item pairs.
    - Use a hash table to count the occurrences of hashed item pairs.
- **Step 2: Candidate Generation and Counting**
    - Generate candidate itemsets from the frequent itemsets obtained in Step 1.
    - Count the occurrences of these candidate itemsets in the dataset.
- **Step 3: Frequent Itemset Filtering**
    - Apply a minimum support threshold to filter out infrequent itemsets.
    - Output the final list of frequent itemsets.
- **Step 4: Parameter Tuning**
    - Experiment with different hash functions and threshold values to observe their impact on performance.

Implementation Tips:

- Optimize hash function selection for efficient pair counting.
- Adjust the threshold for determining frequent itemsets based on dataset characteristics.
- Monitor memory usage and adapt hash table size accordingly.

Lab Experiments to be Performed in This Session:

Execute the PCY algorithm on a transaction dataset to gain insights into its functionality and operation.

NAME: UMANG KIRIT LODAYA

SAP ID: 60009200032

BATCH: D11

PCY ALGORITHM

In [1]:
```
import numpy as np
import pandas as pd

from random import randint
from itertools import combinations
```

In [2]:
```
MIN_SUPPORT = 2
```

In [3]:
```
N = randint(5, 8)
```

In [4]:
```
data = {
#     "TID": ['T' + str(i) for i in range(1, 7)],
#     "ITEMS": [[1, 2, 3], [4, 5], [1, 4, 5], [1, 2, 4], [3,
4, 5], [2, 4, 5]]
    "TID": ['T' + str(i) for i in range(N)],
    "ITEMS": [list(set(randint(1, N) for _ in range(randint(2,
5)))) for _ in range(N)]
}

data = pd.DataFrame(data)
data
```

Out[4]:

	TID	ITEMS
0	T0	[1, 2, 4, 5, 7]
1	T1	[1, 2, 5, 6]
2	T2	[8, 2]
3	T3	[8, 6]
4	T4	[1, 4, 6, 7, 8]
5	T5	[8, 3]
6	T6	[1, 6]
7	T7	[8, 6]

In [5]:
```
S_SUPPORT = {}
for items in data['ITEMS']:
    for item in items:
        S_SUPPORT[item] = S_SUPPORT.get(item, 0) + 1

S_SUPPORT
```

Out[5]:
```
{1: 4, 2: 3, 4: 2, 5: 2, 7: 2, 6: 5, 8: 5, 3: 1}
```

In [6]:
```
S_CANDIDATE = []
for k, v in S_SUPPORT.items():
    if v ≥ MIN_SUPPORT:
        S_CANDIDATE.append(k)

S_CANDIDATE
```

Out[6]:
```
[1, 2, 4, 5, 7, 6, 8]
```

In [7]:
```
D_SUPPORT = {}
for items in data['ITEMS']:
    for i, j in list(combinations(items, 2)):
        key = f"{i},{j}"
        if i in S_CANDIDATE and j in S_CANDIDATE:
            D_SUPPORT[key] = D_SUPPORT.get(key, 0) + 1
        else:
            D_SUPPORT[key] = 0

D_SUPPORT
```

Out[7]:
```
{'1,2': 2,
 '1,4': 2,
 '1,5': 2,
 '1,7': 2,
 '2,4': 1,
 '2,5': 2,
 '2,7': 1,
 '4,5': 1,
 '4,7': 2,
 '5,7': 1,
 '1,6': 3,
 '2,6': 1,
 '5,6': 1,
 '8,2': 1,
 '8,6': 2,
 '1,8': 1,
 '4,6': 1,
 '4,8': 1,
 '6,7': 1,
 '6,8': 1,
 '7,8': 1,
 '8,3': 0}
```

In [8]:
```
D_CANDIDATE = []
for k, v in D_SUPPORT.items():
    if v ≥ MIN_SUPPORT:
        D_CANDIDATE.append(list(map(int, k.split(','))))

D_CANDIDATE
```

Out[8]:
```
[[1, 2], [1, 4], [1, 5], [1, 7], [2, 5], [4, 7], [1, 6], [8,
6]]
```

In [9]:
```
def H(x, y):
    return (x * y) % 10
```

In [10]:
```
BUCKET = {}
for i, j in D_CANDIDATE:
    h = H(i, j)
    BUCKET[h] = BUCKET.get(h, []) + [f"{i},{j}"]

BUCKET
```

Out[10]:
```
{2: ['1,2'],
 4: ['1,4'],
 5: ['1,5'],
 7: ['1,7'],
 0: ['2,5'],
 8: ['4,7', '8,6'],
 6: ['1,6']}
```

In [11]:
```
TABLE = {
    'BIT': [],
    'BUCKET': [],
    'COUNT': [],
    'PAIR': [],
    'CANDIDATE': []
}

for bucket, pairs in BUCKET.items():
    TABLE['BIT'].append(1)
    TABLE['BUCKET'].append(bucket)
    count = 0
    temp = []
    for pair in pairs:
        temp.append(pair.split(','))
        count += D_SUPPORT[pair]


    TABLE['COUNT'].append(count)
    TABLE['PAIR'].append(temp)
    TABLE['CANDIDATE'].append(temp)

TABLE = pd.DataFrame(TABLE)
TABLE
```

Out[11]:

	BIT	BUCKET	COUNT	PAIR	CANDIDATE
0	1	2	2	[[1, 2]]	[[1, 2]]
1	1	4	2	[[1, 4]]	[[1, 4]]
2	1	5	2	[[1, 5]]	[[1, 5]]
3	1	7	2	[[1, 7]]	[[1, 7]]
4	1	0	2	[[2, 5]]	[[2, 5]]
5	1	8	4	[[4, 7], [8, 6]]	[[4, 7], [8, 6]]
6	1	6	3	[[1, 6]]	[[1, 6]]

In [12]:
```
FINAL = []
for pairs in TABLE['CANDIDATE']:
    for pair in pairs:
        FINAL.append(pair)

FINAL
```

Out[12]:
```
[['1', '2'],
 ['1', '4'],
 ['1', '5'],
 ['1', '7'],
 ['2', '5'],
 ['4', '7'],
 ['8', '6'],
 ['1', '6']]
```